Administración de seguridad

Josep Jorba Esteve

P07/M2103/02288



Índex

Introducció

El salto tecnológico de los sistemas de escritorio aislados, hasta los sistemas actuales integrados en redes locales e Internet, ha traído una nueva dificultad a las tareas habituales del administrador: el control de la seguridad de los sistemas.

La seguridad es un campo complejo, en el cual se mezclan técnicas de análisis con otras de detección o de prevención de los posibles ataques. Como el análisis de factores "psicológicos", respecto el comportamiento de los usuarios del sistema o las posibles intenciones de los atacantes.

Los ataques pueden provenir de muchas fuentes y afectar desde a una aplicación o servicio hasta a algún usuario, o a todos, o al sistema informático entero.

Los posibles ataques pueden cambiar el comportamiento de los sistemas, incluso hacerlos caer (inutilizarlos), o dar una falsa impresión de seguridad, que puede ser difícilmente detectable. Podemos encontrarnos con ataques de autentificación (obtener acceso por parte de programas o usuarios previamente no habilitados), escuchas (redirigir o pinchar los canales de comunicación y los datos que circulan), o sustitución (sustituir programas, máquinas o usuarios por otros, sin que se noten los cambios).

Nota

La seguridad absoluta no existe. Una falsa impresión de seguridad puede ser tan perjudicial como no tenerla. El área de la seguridad es muy dinámica, y hay que mantener actualizados constantemente los conocimientos.

Una idea clara que hay que tener en mente es que es imposible poder conseguir una seguridad al 100%.

Las técnicas de seguridad son un arma de doble filo, que fácilmente pueden darnos una falsa impresión de control del problema. La seguridad actual es un problema amplio, complejo, y lo que es más importante, dinámico. Nunca podemos esperar o decir que la seguridad está garantizada, sino que con bastante probabilidad será una de las áreas a la cual el administrador tendrá que dedicar más tiempo y mantener actualizados sus conocimientos sobre el tema.

En esta unidad examinaremos algunos tipos de ataques con los que podemos encontrarnos, cómo podemos verificar y prevenir partes de la seguridad local, y también en entornos de red. Asimismo, examinaremos técnicas de detección de intrusos y algunas herramientas básicas que nos pueden ayudar en el control de la seguridad.

También es preciso mencionar que en esta unidad sólo podemos hacer una mera introducción a algunos de los aspectos que intervienen en la seguridad de hoy en día. Para cualquier aprendizaje real con más detalle, se recomienda consultar la bibliografía disponible, así como los manuales asociados a los productos o herramientas comentados.

1. Tipos y métodos de los ataques

La seguridad computacional en administración puede ser entendida como el proceso que ha de permitir al administrador del sistema prevenir y detectar usos no autorizados de éste. Las medidas de prevención ayudan a parar los intentos de usuarios no autorizados (los conocidos como intrusos), para acceder a cualquier parte del sistema. La detección ayuda a descubrir cuándo se produjeron estos intentos o, en el caso de llevarse a cabo, establecer las barreras para que no se repitan y poder recuperar el sistema si éste ha sido quebrantado.

Los intrusos (también conocidos coloquialmente como *hackers, crackers,* 'atacantes' o 'piratas'...) normalmente desean obtener control sobre el sistema, ya sea para causar malfuncionamiento, corromper el sistema o sus datos, ganar recursos en la máquina, o simplemente utilizarlo para lanzar ataques contra otros sistemas, y así proteger su verdadera identidad y ocultar el origen real de los ataques. También está la posibilidad de examinar la información (o robarla) del sistema, el puro espionaje de las acciones del sistema o causar daños físicos en la máquina, ya sea formatear el disco, cambiar datos, borrar o modificar software crítico, etc.

Con respecto a los intrusos, hay que establecer algunas diferencias, que no suelen estar muy claras en los términos coloquiales. Normalmente nos referimos a *hacker* [Him01], como aquella persona con grandes conocimientos en informática, más o menos apasionada por los temas de programación y seguridad informática, y que, normalmente sin finalidad malévola, utiliza sus conocimientos para protegerse a sí mismo, o a terceros, introducirse en redes para demostrar sus fallos de seguridad, y, en algunos casos, como reconocimiento de sus habilidades.

Un ejemplo sería la propia comunidad GNU/Linux, que debe mucho a sus *hackers*, ya que hay que entender el término *hacker* como experto en unos temas (más que como intruso en la seguridad).

Por otro lado encontraríamos a los *crackers*. Aquí es cuando se utiliza el término de manera más o menos despectiva, hacia aquellos que utilizan sus habilidades para corromper (o destrozar) sistemas, ya sea sólo por fama propia, por motivos económicos, por ganas de causar daño o simplemente por molestar; por motivos de espionaje tecnológico, actos de ciberterrorismo, etc. Asimismo, se habla de *hacking* o *cracking*, cuando nos referimos a técnicas de estudio, detección y protección de la seguridad, o por el contrario, a técnicas destinadas a causar daño rompiendo la seguridad de los sistemas.

Desafortunadamente, obtener acceso a un sistema (ya sea desprotegido o parcialmente seguro) es bastante más fácil de lo que parece. Los intrusos descubren permanentemente nuevas vulnerabilidades (llamadas a veces 'agujeros', o *exploits*), que les permiten introducirse en las diferentes capas de software. La complejidad cada vez mayor del software (y del hardware), hace que aumente la dificultad para testear de manera razonable la seguridad de los sistemas informáticos. El uso habitual de los sistemas GNU/Linux en red, ya sea la propia Internet o en redes propias con tecnología TCP/IP como las intranets, nos lleva a exponer nuestros sistemas, como víctimas, a ataques de seguridad. [Bur02][Fen02][Line]

Lo primero que hay que hacer es romper el mito de la seguridad informática: simplemente no existe. Lo que sí que podemos conseguir es un cierto nivel de seguridad que nos haga sentirnos seguros dentro de ciertos parámetros. Pero como tal, sólo es una percepción de seguridad, y como todas las percepciones, puede ser falsa y de ello podemos darnos cuenta en el último momento, cuando ya tengamos nuestros sistemas afectados. La conclusión lógica es que la seguridad informática exige un esfuerzo importante de constancia, realismo y aprendizaje prácticamente diario.

Tenemos que ser capaces de establecer en nuestros sistemas unas políticas de seguridad que nos permitan prevenir, identificar y reaccionar ante los posibles ataques. Y tener presente que la sensación que podamos tener de seguridad no es más que eso, una sensación. Por lo tanto, no hay que descuidar ninguna de las políticas implementadas y hay que mantenerlas al día, así como nuestros conocimientos del tema.

Los posibles ataques son una amenaza constante a nuestros sistemas y pueden comprometer su funcionamiento, así como los datos que manejamos; ante todo lo cual, siempre tenemos que definir una cierta política de requisitos de seguridad sobre nuestros sistemas y datos. Las amenazas que podemos sufrir podrían afectar a los aspectos siguientes:

Nota

Las amenazas afectan a la confidencialidad, integridad o accesibilidad de nuestros sistemas.

- **Confidencialidad**: la información debe ser accesible sólo a aquellos que estén autorizados; estamos respondiendo a la pregunta: ¿quién podrá acceder a la misma?

- **Integridad**: la información sólo podrá ser modificada por aquellos que estén autorizados: ¿qué se podrá hacer con ella?

- **Accesibilidad**: la información tiene que estar disponible para quienes la necesiten y cuando la necesiten, si están autorizados: ¿de qué manera, y cuándo se podrá acceder a ella?

Pasemos a mencionar una cierta clasificación (no exhaustiva) de los tipos de ataques habituales que podemos padecer:

- **Autentificación**: ataques en los que se falsifica la identidad del participante de manera que obtiene acceso a programas o servicios de los que no disponía en un principio.

- **Intercepción** (o escucha): mecanismo por el cual se interceptan datos, por parte de terceros, cuando éstos no estaban dirigidos a ellos.

- **Falsificación** (o reemplazo): sustitución de algunos de los participantes –ya sea máquinas, software o datos– por otros falsos.

- **Robo** de recursos: uso de nuestros recursos sin autorización.

- O, simplemente, **vandalismo**: después de todo, suele ser bastante común la existencia de mecanismos que permiten interferir con el funcionamiento adecuado del sistema o servicios y causar molestias parciales, o el paro o cancelación de recursos.

Los métodos utilizados y las técnicas precisas pueden variar mucho (es más, cada día se crean novedades), y nos obligan, como administradores, a estar en contacto permanente con el campo de la seguridad para conocer a qué nos podemos enfrentar diariamente.

Para cada uno de los tipos de ataques mencionados, normalmente se pueden utilizar uno o más métodos de ataque, que a su vez pueden provocar también uno o más de los tipos de ataques.

Respecto a dónde se produzca el ataque, hemos de tener claro qué puede hacerse o cuál será el objetivo de los métodos:

Nota

Los ataques pueden tener finalidades destructivas, inhabilitadoras o de espionaje, de nuestros componentes, ya sea hardware, software o sistemas de comunicación.

- **Hardware**: a este respecto, la amenaza está directamente sobre la accesibilidad, ¿qué podrá hacer alguien que tenga acceso al hardware? En este caso normalmente necesitaremos medidas "físicas", como controles de seguridad para acceder a los locales donde estén las máquinas para evitar problemas de robo o rotura del equipo con el fin de eliminar su servicio. También puede comprometerse la confidencialidad y la integridad si el acceso físico a las máquinas permite utilizar algunos de sus dispositivos como las disqueteras, o el arranque de las máquinas, o el acceso a cuentas de usuario que podrían estar abiertas.

- **Software**: si la accesibilidad se ve comprometida en un ataque, puede haber borrado o inutilización de programas, denegando el acceso. En caso de confidencialidad, puede provocar copias no autorizadas de software. En integridad podría alterarse el funcionamiento por defecto del programa, para que éste falle en algunas situaciones o bien para que realice tareas que puedan ser interesantes de cara al atacante, o simplemente comprometer la integridad de los datos de los programas: hacerlos públicos, alterarlos, o simplemente robarlos.

- **Datos**: ya sean estructurados, como en los servicios de base de datos, o gestión de versiones (como cvs), o simples archivos. Mediante ataques que amenacen la accesibilidad, pueden ser destruidos o eliminados, denegando así el acceso a los mismos. En el caso de la confidencialidad, estaríamos permitiendo lecturas no autorizadas y la integridad se vería afectada cuando se produzcan modificaciones o creación de nuevos datos.

- **Canal de comunicación** (en la red, por ejemplo): para los métodos que afecten a la accesibilidad, nos puede provocar destrucción o eliminación de mensajes, e impedir el acceso a la red. En confidencialidad, lectura y observación del tráfico de mensajes, desde o hacia la máquina. Y respecto a la integridad, cualquier modificación, retardo, reordenación, duplicación o falsificación de los mensajes entrantes y/o salientes.

1.1. Técnicas utilizadas en los ataques

Los métodos utilizados son múltiples y pueden depender de un elemento (hardware o software) o de la versión de éste. Por lo tanto, hay que mantener actualizado el software para las correcciones de seguridad que vayan apareciendo, y seguir las indicaciones del fabricante o distribuidor para proteger el elemento.

A pesar de ello, normalmente siempre hay técnicas o métodos de “moda”, del momento actual. Algunas breves indicaciones de de estas técnicas de ataque (de hoy en día) son:

Nota

Las técnicas de los ataques son muy variadas y evolucionan constantemente en lo que a los detalles usados se refiere.

- ***Bug exploits***: o explotación de errores o agujeros [CERb] [Ins][San], ya sea de un hardware, software, servicio, protocolo o del propio sistema operativo (por ejemplo, en el *kernel*), y normalmente de alguna de las versiones de éstos en concreto. Normalmente, cualquier elemento informático es más o menos propenso a errores en su concepción, o simplemente a cosas que no se han tenido en cuenta o previsto. Periódicamente, se descubren agujeros (a veces se denominan *holes, exploits*, o simplemente *bugs*...), que pueden ser aprovechados por un atacante para romper la seguridad de los sistemas. Suelen utilizarse o bien técnicas de ataque genéricas, como las que se explican a continuación, o bien técnicas particulares para el elemento afectado. Cada elemento afectado tendrá un responsable –ya sea fabricante, desarrollador, distribuidor o la comunidad GNU/Linux– de producir nuevas versiones o parches para tratar estos problemas. Nosotros, como administradores, tenemos la responsabilidad de estar informados y de mantener una política de actualización responsable para evitar los riesgos potenciales de estos ataques. En caso de que no haya soluciones disponibles, también podemos estudiar la posibilidad de utilizar alternativas al elemento, o bien inhabilitarlo hasta que tengamos soluciones.

- **Virus**: programa normalmente anexo a otros y que utiliza mecanismos de autocopia y transmisión. Son habituales los virus anexados a programas ejecutables, a mensajes de correo electrónico, o incorporados en documentos o programas que permiten algún lenguaje de macros (no verificado). Son quizás la mayor plaga de seguridad de hoy en día.

  Los sistemas GNU/Linux están protegidos casi totalmente contra estos mecanismos por varias razones: en los programas ejecutables, tienen un acceso muy limitado al sistema, en particular a la cuenta del usuario. Con excepción del usuario *root*, con el que hay que tener mucho cuidado con lo que ejecuta. El correo no suele utilizar lenguajes de macros no verificados (como en el caso de Outlook y Visual Basic Script en Windows, que es un agujero de entrada de virus), y en el caso de los documentos, estamos en condiciones parecidas, ya que no soportan lenguajes de macros no verificados (como el VBA en Microsoft Office).

  En todo caso, habrá que prestar atención a lo que pueda pasar en un futuro, ya que podrían surgir algunos virus específicos para GNU/Linux aprovechando algunos *bugs* o *exploits*. Un punto que sí que hay que tener en cuenta es el de los sistemas de correo, ya que si bien nosotros no generaremos virus, sí que podemos llegar a transmitirlos; por ejemplo, si nuestro sistema funciona como *router* de correo, podrían llegar mensajes con virus que podrían ser enviados a otros. Aquí se puede implementar alguna política de detección y filtrado de virus. Otra forma de azote de plagas que podría entrar dentro de la categoría de virus son los mensajes de *spam*, que si bien no suelen ser utilizados como elementos atacantes, sí que podemos considerarlos como problemas por su "virulencia" de aparición, y el coste económico que pueden causar (pérdidas de tiempo y recursos).

- ***Worm*** (o 'gusano'): normalmente se trata de un tipo de programas que aprovechan algún agujero del sistema para realizar ejecuciones de código sin permiso. Suelen ser utilizados para aprovechar recursos de la máquina, como el uso de CPU, bien cuando se detecta que el sistema no funciona o no está en uso, o bien si son malintencionados, con el objetivo de robar recursos o bien utilizarlos para parar o bloquear el sistema. También suelen utilizar técnicas de transmisión y copia.

- ***Trojan horse*** (o 'caballos de Troya', o 'troyanos'): programas útiles que incorporan alguna funcionalidad, pero ocultan otras que son las utilizadas para obtener información del sistema o comprometerlo. Un caso particular puede ser el de los códigos de tipo móvil en aplicaciones web, como los Java, JavaScript o ActiveX; éstos normalmente piden su consentimiento para ejecutarse (ActiveX en Windows), o tienen modelos limitados de lo que pueden hacer (Java, JavaScript). Pero como todo software, también tienen agujeros y son un método ideal para transmitir troyanos.

- ***Back door*** (o *trap door*, 'puerta trasera'): método de acceso a un programa escondido que puede utilizarse con fines de otorgar acceso al sistema o a los datos manejados sin que lo conozcamos. Otros efectos pueden ser cambiar la configuración del sistema, o permitir introducir virus. El mecanismo usado puede ser desde venir incluidos en algún software común, hasta en un troyano.

- **Bombas lógicas**: programa incrustado en otro, que comprueba que se den algunas condiciones (temporales, acciones del usuario, etc.), para activarse y emprender acciones no autorizadas.

- ***Keyloggers***: programa especial que se dedica a secuestrar las interacciones con el teclado, y/o ratón, del usuario. Pueden ser programas individuales o bien troyanos incorporados en otros programas.

  Normalmente, necesitarían introducirse en un sistema abierto al que se dispusiese de acceso (aunque cada vez más pueden venir incorporados en troyanos que se instalen). La idea es captar cualquier introducción de teclas, de manera que se capturen contraseñas (por ejemplo, las bancarias), interacción con aplicaciones, sitios visitados por la red, formularios rellenados, etc.

- ***Scanner*** (escaneo de puertos): más que un ataque, sería un paso previo, que consistiría en la recolección de posibles objetivos. Básicamente, consiste en utilizar herramientas que permitan examinar la red en busca de máquinas con puertos abiertos, sean TCP, UDP u otros protocolos, los cuales indican la presencia de algunos servicios. Por ejemplo, escanear máquinas buscando el puerto 80 TCP, indica la presencia de servidores web, de los cuales podemos obtener información sobre el servidor y la versión que utilizan para aprovecharnos de vulnerabilidades conocidas.

- ***Sniffers*** ('husmeadores'): permiten la captura de paquetes que circulan por una red. Con las herramientas adecuadas podemos analizar comportamientos de máquinas: cuáles son servidores, clientes, qué protocolos se utilizan y en muchos casos obtener contraseñas de servicios no seguros. En un principio, fueron muy utilizados para capturar contraseñas de telnet, rsh, rcp, ftp... servicios no seguros que no tendrían que utilizarse (usar en su lugar las versiones seguras: ssh, scp, sftp). Tanto los *sniffers* (como los *scanners*) no son necesariamente una herramienta de ataque, ya que también pueden servir para analizar nuestras redes y detectar fallos, o simplemente analizar nuestro tráfico. Normalmente, tanto las técnicas de *scanners* como las de *sniffers* suelen utilizarse por parte de un intruso con el objetivo de encontrar las vulnerabilidades del sistema, ya sea para conocer datos de un sistema desconocido (*scanners*), o bien para analizar la interacción interna (*sniffer*).

- ***Hijacking*** (o 'secuestro'): son técnicas que intentan colocar una máquina de manera que intercepte o reproduzca el funcionamiento de algún servicio en otra máquina que ha pinchado la comunicación. Suelen ser habituales los casos para correo electrónico, transferencia de ficheros o web. Por ejemplo, en el caso web, se puede capturar una sesión y reproducir lo que el usuario está haciendo, páginas visitadas, interacción con formularios, etc.

- ***Buffer overflows***: técnica bastante compleja que aprovecha errores de programación en las aplicaciones. La idea básica es aprovechar desbordamientos (*overflows*) en *buffers* de la aplicación, ya sean colas, *arrays*, etc. Si no se controlan los límites, un programa atacante puede generar un mensaje o dato más grande de lo esperado y provocar fallos. Por ejemplo, muchas aplicaciones C con *buffers* mal escritos, en *arrays*, si sobrepasamos el límite podemos provocar una sobreescritura del código del programa, causando malfuncionamiento o caída del servicio o máquina. Es más, una variante más compleja permite incorporar en el ataque trozos de programa (compilados C o bien *shell scripts*), que pueden permitir la ejecución de cualquier código que el atacante quiera introducir.

- ***Denial of Service*** ('ataque DoS'): este tipo de ataque provoca que la máquina caiga o que se sobrecarguen uno o más servicios, de manera que no sean utilizables. Otra técnica es la DDoS (Distributed DoS), que se basa en utilizar un conjunto de máquinas distribuidas para que produzcan el ataque o sobrecarga de servicio. Este tipo de ataques se suelen solucionar con actualizaciones del software, ya que normalmente se ven afectados aquellos servicios que no fueron pensados para una carga de trabajo determinada y no se controla la saturación. Los ataques DoS y DDoS son bastante utilizados en ataques a sitios web, o servidores DNS, que se ven afectados por vulnerabilidades de los servidores, por ejemplo, de versiones concretas de Apache o BIND. Otro aspecto que cabe tener en cuenta es que nuestro sistema también podría ser usado para ataques de tipo DDoS, mediante control, ya sea de un *backdoor* o un troyano.

  Un ejemplo de este ataque (DoS) bastante sencillo es el conocido como SYN flood, que trata de generar paquetes TCP que abren una conexión, pero ya no hacen nada más con ella, simplemente la dejan abierta; esto gasta recursos del sistema en estructuras de datos del *kernel*, y recursos de conexión por red. Si se repite este ataque centenares o miles de veces, se consigue ocupar todos los recursos sin utilizarlos, de modo que cuando algunos usuarios quieran utilizar el servicio, les sea denegado porque los recursos están ocupados. Otro caso conocido es el correo *bombing*, o simplemente reenvío de correo (normalmente con emisor falso) hasta que se saturan las cuentas de correo, el sistema de correo cae, o se vuelve tan lento que es inutilizable. Estos ataques son en cierta medida sencillos de realizar con las herramientas adecuadas, y no tienen una solución fácil, ya

Nota

SYN flood, ver: http://www.cert.org/advisories/CA-1996-21.html

E-mail bombing, ver: http://www.cert.org/tech tips/email_bombing_spamming.html

que se aprovechan del funcionamiento interno de los protocolos y servicios; en estos casos tenemos que tomar medidas de detección y control posterior.

- ***Spoofing***: las técnicas de *spoofing* engloban varios métodos (normalmente muy complejos) de falsificar tanto la información, como los participantes en una transmisión (origen y/o destino). Algunos *spoofing* son los de:

  – IP spoofing, falsificación de una máquina, permitiendo que genere tráfico falso o escuche tráfico que iba dirigido a otra máquina. Combinado con otros ataques puede saltarse incluso protección de *firewalls*.

  – ARP spoofing, técnica compleja (utiliza un DDoS), que intenta falsificar las direcciones de fuentes y destinatarios en una red, mediante ataques de las cachés de ARP, que poseen las máquinas, de manera que se sustituyan las direcciones reales por otras en varios puntos de una red. Esta técnica permite saltarse todo tipo de protecciones, incluidos *firewalls*, pero no es una técnica sencilla.

  – E-mail, es quizás el más sencillo. Se trata de generar contenidos de correos falsos, tanto por el contenido como por la dirección de origen. En este tipo son bastante utilizadas las técnicas de lo que se denomina ingeniería social, que básicamente intenta engañar de una forma razonable al usuario, un ejemplo clásico son correos falsos del administrador de sistema, o bien, por ejemplo, del banco donde tenemos nuestra cuenta corriente, mencionando que ha habido problemas en las cuentas y que se tiene que enviar información confidencial, o la contraseña anterior para solucionarlo, o pidiendo que la contraseña se cambie por una concreta. Sorprendentemente, esta técnica (también conocida como *phising*) consigue engañar a un número considerable de usuarios. Incluso con (ingeniera social de) métodos sencillos: algún *cracker* famoso comentaba que su método preferido era el teléfono. Como ejemplo, exponemos el caso de una empresa de certificación (*verisign*), de la que los *crackers* obtuvieron la firma privada de software Microsoft con sólo realizar una llamada diciendo que llamaban de parte de la empresa, que se les había presentado un problema y volvían a necesitar su clave. Resumiendo, unos niveles altos de seguridad informática se pueden ir al traste por una simple llamada telefónica, o un correo que un usuario malinterprete.

Nota

Ver el caso de Microsoft en: http://www.computerworld.com/softwaretopics/os/windows/story/0,10801,59099,00.html

- ***SQL injection***: es una técnica orientada a bases de datos, y servidores web en particular, que se aprovecha generalmente de programación incorrecta de formularios web, en los cuales no se ha controlado correctamente la información que se proporciona. No se determina que la información de entrada sea del tipo correcto (fuertemente tipada respecto a lo que se espera), o no se controlan qué tipo o caracteres literales se introducen. La técnica se aprovecha de que los literales obtenidos por los formularios (por ejemplo web, aunque los ataques pueden sufrirse desde cualquier API que permita acceso a base de datos,

por ejemplo php o perl) son utilizados directamente para construir las consultas (en SQL), que atacarán una determinada base de datos (a la que en principio no se tiene acceso directo). Normalmente, si existen las vulnerabilidades y poco control de los formularios, se puede inyectar código SQL en el formulario, de manera que se construyan consultas SQL que proporcionen la información buscada. En casos drásticos podría obtenerse la información de seguridad (usuarios y contraseñas de la base de datos), o incluso tablas o la base de datos entera, cuando no podrían producirse pérdidas de información, o borrados intencionados de datos. En particular esta técnica en ambientes web puede ser grave, debido a las leyes que protegen la privacidad de datos personales que un ataque de este tipo puede provocar. En este caso, más que una cuestión de seguridad de sistema, supone un problema de programación y control con tipado fuerte de los datos esperados en la aplicación, además del adecuado control de conocimiento de vulnerabilidades presentes del software usado (base de datos, servidor web, API como php, perl...).

- ***Cross-side scripting*** (o XSS): otra problemática relacionada con ambientes web, en particular con alteraciones del código html, y/o *scripts* que puede obtener un usuario visualizando un determinado sitio web, que puede ser alterado dinámicamente. Se aprovecha generalmente de los errores a la hora de validar código HTML (todos los navegadores tienen problemas con esto, por la propia definición de HTML, que permite leer prácticamente cualquier código HTML por incorrecto que sea). En algunos casos, la utilización de vulnerabilidades puede ser directa mediante *scripts* en la página web, pero normalmente los navegadores tienen buen control de éstos. Por otra parte, indirectamente hay técnicas que permiten insertar código de *script*, o bien mediante acceso a las *cookies* del usuario desde el navegador, o mediante la alteración del proceso por el que se redirecciona una página web a otra. También hay técnicas mediante *frames*, que permiten redirigir el html que se esté viendo, o colgar directamente el navegador. En particular pueden ser vulnerables los motores de búsqueda de los sitios web, que pueden permitir ejecución de código de *script*. En general son ataques con diversas técnicas complejas, pero con objetivo de capturar información como *cookies*, que pueden ser usadas en sesiones, y permitir así sustituir a una determinada persona mediante redirecciones de sitios web, u obtener información suya. Otra vez desde la perspectiva de sistema, es más una cuestión de software en uso. Hay que controlar y conocer vulnerabilidades detectadas en navegadores (y aprovechar los recursos que éstos ofrecen para evitar estas técnicas), y controlar el uso de software (motores de búsqueda empleados, versiones del servidor web, y API utilizadas en los desarrollos).

Algunas recomendaciones generales (muy básicas) para la seguridad, podrían ser:

- Controlar un factor problemático: los usuarios. Uno de los factores que puede afectar más a la seguridad es la confidencialidad de las contraseñas, y ésta se ve afectada por el comportamiento de los usuarios; esto facilita a posibles ata-

cantes las acciones desde dentro del propio sistema. La mayoría de los ataques suelen venir de dentro del sistema, o sea, una vez el atacante ha ganado acceso al sistema.

- Entre los usuarios, está aquel que es un poco olvidadizo (o indiscreto), que bien olvida la contraseña cada dos por tres, la menciona en conversaciones, la escribe en un papel que olvida, o que está junto (o pegado) al ordenador o sobre la mesa de trabajo, o que simplemente la presta a otros usuarios o conocidos. Otro tipo es el que coloca contraseñas muy predecibles, ya sea su mismo id de usuario, su nombre, su DNI, el nombre de su novia, el de su madre, el de su perro, etc., cosas que con un mínimo de información pueden encontrarse fácilmente. Otro caso son los usuarios normales con un cierto conocimiento, que colocan contraseñas válidas, pero siempre hay que tener en cuenta que hay mecanismos que pueden encontrarlas (*cracking de passwords, sniffing, spoofing*...). Hay que establecer una cierta "cultura" de seguridad entre los usuarios, y mediante técnicas obligarles a que cambien las contraseñas, no utilicen palabras típicas, las contraseñas deben ser largas (tener más de 2 o 3 caracteres), etc. Últimamente, en muchas empresas e instituciones se está implantando la técnica de hacer firmar un contrato al usuario de manera que se le obliga a no divulgar la contraseña o cometer actos de vandalismo o ataques desde su cuenta (claro que esto no impide que otros lo hagan por él).

- No utilizar ni ejecutar programas de los que no podamos garantizar su origen. Normalmente, muchos distribuidores utilizan mecanismos de comprobación de firmas para verificar que los paquetes de software son tales, como por ejemplo las sumas md5 (comando md5sum) o la utilización de firmas GPG [Hatd](comando gpg). El vendedor o distribuidor provee una suma md5 de su archivo (o imagen de CD), y podemos comprobar la autenticidad de éste. Últimamente, en las distribuciones se están usando tanto firmas para paquetes individuales, como las firmas para los repositorios de paquetes, como mecanismo para asegurar la fiabilidad del proveedor.

- No utilizar usuarios privilegiados (como *root*) para el trabajo normal de la máquina; cualquier programa (o aplicación) tendría los permisos para acceder a cualquier parte.

- No acceder remotamente con usuarios privilegiados ni ejecutar programas que puedan tener privilegios. Y más si no conocemos, o no hemos comprobado, los niveles de seguridad del sistema.

- No utilizar elementos que no sabemos cómo actúan ni intentar descubrirlo a base de repetidas ejecuciones.

Estas medidas pueden ser poco productivas, pero si no hemos asegurado el sistema, no podemos tener ningún control sobre lo que puede pasar, y aun así,

nadie asegura que no se pueda colar algún programa malicioso que burle la seguridad si lo ejecutamos con los permisos adecuados. O sea, que en general hemos de tener mucho cuidado con todo este tipo de actividades que supongan accesos y ejecución de tareas de formas más o menos privilegiadas.

1.2. Contramedidas

Respecto a las medidas que se pueden tomar sobre los tipos de ataques presentados, podemos encontrar algunas preventivas, y de detección de lo que sucede en nuestros sistemas.

Veamos algunos tipos de medidas que podríamos tomar en los ámbitos de prevención y detección de intrusos (se mencionan herramientas útiles, algunas las examinaremos más adelante):

- ***Password cracking***: en ataques de fuerza bruta para romper las contraseñas, suele ser habitual intentar obtener acceso por *logins* de forma repetida; si se consigue entrar, la seguridad del usuario ha sido comprometida y se deja la puerta abierta a otros tipos de ataques como, por ejemplo, los *backdoor*, o simplemente la destrucción de la cuenta. Para prevenir este tipo de ataques, hay que reforzar la política de contraseñas, pidiendo una longitud mínima y cambios de contraseña periódicos. Una cosa que hay que evitar es el uso de palabras comunes en las contraseñas: muchos de estos ataques se hacen mediante la fuerza bruta, con un fichero de diccionario (con palabras en el idioma del usuario, términos comunes, argot, etc.). Este tipo de contraseñas serán las primeras en caer. También puede ser fácil obtener información del atacado, como nombres, DNI o su dirección, y usar estos datos para probar las contraseñas. Por todo lo cual, tampoco se recomiendan contraseñas con DNI, nombres (propios o de familiares, etc.), direcciones, etc. Una buena elección suele ser elegir contraseñas de entre 6 a 8 caracteres como mínimo, con contenido de caracteres alfabéticos, numéricos y algún carácter especial.

  Aunque la contraseña esté bien elegida, ésta puede ser insegura si se utiliza en servicios no seguros. Por lo tanto, se recomienda reforzar los servicios mediante técnicas de encriptación que protejan las contraseñas y los mensajes. Y por el contrario, evitar (o no utilizar) todos aquellos servicios que no soporten encriptación, y consecuentemente, susceptibles de ser atacados con métodos, por ejemplo de *sniffers*; entre éstos podríamos incluir servicios telnet, ftp, rsh, rlogin (entre otros).

- ***Bug exploits:*** evitar disponer de programas que no se utilicen, sean antiguos, o no se actualicen (por estar obsoletos). Aplicar los últimos parches y actualizaciones que estén disponibles, tanto para las aplicaciones como para el sistema operativo. Probar herramientas que detecten vulnerabilidades. Mantenerse al día de las vulnerabilidades que se vayan descubriendo.

Nota

Ver parches para el sistema operativo en:
http://www.debian.org/security
http://www.redhat.com//security
http://fedoraproject.org/wiki/Security

- **Virus**: utilizar mecanismos o programas antivirus, sistemas de filtrado de mensajes sospechosos, evitar la ejecución de sistemas de macros (que no se puedan verificar). No hay que minimizar los posibles efectos de los virus, cada día se perfeccionan más, y técnicamente es posible realizar virus simples que puedan desactivar redes en cuestión de minutos (sólo hay que ver algunos de los virus recientes del mundo Windows).

- ***Worm*** (o gusano): controlar el uso de nuestras máquinas o usuarios en horas no previstas, y el control del tráfico de salida y/o entrada.

- ***Trojan horse*** (o caballos de Troya, o troyanos): verificar la integridad de los programas periódicamente, mediante mecanismos de suma o firmas. Detección de tráfico anómalo de salida o entrada al sistema. Utilizar *firewalls* para bloquear tráfico sospechoso. Una versión bastante peligrosa de los troyanos la forman los *rootkits* (comentados más adelante), que realizan más de una función gracias a un conjunto variado de herramientas. Para la verificación de la integridad, podemos utilizar mecanismos de sumas como (md5 o gpg) o herramientas que automatizan este proceso, como Tripwire o AIDE.

- ***Backdoor*** (o *trap door*, puerta trasera): hay que obtener de los proveedores o vendedores del software la certificación de que éste no contiene ningún tipo de *backdoor* escondido no documentado, y por supuesto aceptar el software proveniente sólo de sitios que ofrezcan garantías. Cuando el software sea de terceros, o de fuentes que podrían haber modificado el software original, muchos fabricantes (o distribuidores) integran algún tipo de verificación de software basado en códigos de suma o firmas digitales (tipo md5 o gpg) [Hatd]. Siempre que éstas estén disponibles, sería útil verificarlas antes de proceder a la instalación del software. También puede probarse el sistema intensivamente, antes de colocarlo como sistema de producción.

  Otro problema puede consistir en la alteración del software a posteriori. En este caso pueden ser también útiles los sistemas de firmas o sumas para crear códigos sobre software ya instalado y controlar que no se produzcan cambios en software vital. O bien copias de seguridad, con las que podemos hacer comparaciones para detectar cambios.

- **Bombas lógicas**: en este caso suelen ocultarse tras activaciones por tiempo o por acciones del usuario. Podemos verificar que no existan en el sistema trabajos no interactivos introducidos de tipo crontab, at, y otros procesos (por ejemplo, de tipo nohup), que dispongan de ejecución periódica, o estén en ejecución en segundo plano desde hace mucho tiempo (comandos w, jobs). En todo caso, podrían utilizarse medidas preventivas que impidieran trabajos no interactivos a los usuarios, o que solamente los permitiesen a aquellos que lo necesitasen.

Nota

Sobre vulnerabilidades, una buena herramienta es Nessus. Para descubrir nuevas vulnerabilidades, ver CERT en: http://www.cert.org/advisories/

- ***Keyloggers y rootkits***: en este caso habrá algún proceso intermediario que intentará capturar nuestras pulsaciones de teclas y las almacenará en algún lugar. Habrá que examinar situaciones donde aparezca algún proceso extraño perteneciente a nuestro usuario, o bien detectar si tenemos algún fichero abierto con el que no estemos trabajando directamente (por ejemplo, podría ser de ayuda lsof, ver man), o bien conexiones de red, si se tratase de keylogger con envío externo. Para probar un funcionamiento muy básico de un keylogger muy sencillo, puede verse el comando de sistema *script* (ver man *script*). El otro caso, el *rootkit* (que suele incluir también algún keylogger) suele ser un pack de unos cuantos programas con varias técnicas, y permite al atacante, una vez entra en una cuenta, utilizar diversos elementos como un keylogger, backdoors, troyanos (sustituyendo a comandos del sistema), etc., con tal de obtener información y puertas de entrada al sistema, muchas veces se acompaña de programas que realizan limpieza de los *logs,* para eliminar las pruebas de la intrusión. Un caso particularmente peligroso lo forman los *rootkits*, que se usan o vienen en forma de módulos de *kernel*, que les permite actuar a nivel de *kernel*. Para su detección, sería necesario controlar que no haya tráfico externo que salga hasta una cierta dirección. Una herramienta útil para verificar los *rootkits* es *chrootkit*.

- **Escáner** (escaneo de puertos): los escaners suelen lanzar sobre uno o más sistemas bucles de escaneo de puertos conocidos para detectar los que quedan abiertos y aquellos servicios que están funcionando (y obtener información de las versiones de los servicios) y que podrían ser susceptibles de ataques.

Nota

La herramienta chrootkit puede encontrarse en: http://www.chrootkit.org

- ***Sniffers*** (husmeadores): evitar intercepciones e impedir así la posibilidad de que se introduzcan escuchas. Una técnica es la construcción hardware de la red, que puede dividirse en segmentos para que el tráfico sólo circule por la zona que se va a utilizar, poner *firewalls* para unir estos segmentos y poder controlar el tráfico de entrada y salida. Usar técnicas de encriptación para que los mensajes no puedan ser leídos e interpretados por alguien que escuche la red. Para el caso tanto de escáneres como de *sniffers*, podemos utilizar herramientas como Whireshark [Wir] (antiguo Ethereal) y Snort, para realizar comprobaciones sobre nuestra red, o para el port scanning Nmap. En el caso de los *sniffers*, éstos pueden ser detectados en la red mediante la búsqueda de máquinas en modo Ethernet promiscuo (están a la escucha de cualquier paquete que circula), normalmente, la tarjeta de red sólo captura el tráfico que va hacia ella (o de tipo *broadcast* o *multicast*).

- ***Hijacking*** (o 'secuestro'): implementar mecanismos de encriptación en los servicios, requerir autentificación y, si es posible, que esta autentificación se renueve periódicamente. Controlar el tráfico entrante o saliente por *firewalls*. Monitorizar la red para detectar flujos de tráfico sospechosos.

- ***Buffer overflows***: suelen ser comunes como *bugs* o agujeros del sistema, suelen solucionarse por actualización del software. En todo caso, pueden observarse por *logs* situaciones extrañas de caída de servicios que deberían estar funcionando. También pueden maximizarse los controles de procesos y accesos a recursos para aislar el problema cuando se produzca en entornos de acceso controlado, como el que ofrece SELinux ver más adelante en el módulo.

- ***Denial of Service*** ('ataque DoS'), y otros como SYN flood, o correos bombing tomar medidas de bloqueo de tráfico innecesario en nuestra red (por ejemplo, por medio de *firewalls*). En aquellos servicios que se pueda, habrá que controlar tamaños de buffer, número de clientes por atender, timeouts de cierre de conexiones, capacidades del servicio, etc.

- ***Spoofing***: a) IP spoofing, b) ARP spoofing, c) correo electrónico. Estos casos necesitan una fuerte encriptación de los servicios, control por *firewalls*, mecanismos de autentificación basados en varios aspectos (por ejemplo, no basarse en la IP, si ésta pudiera verse comprometida), se pueden implementar mecanismos que controlen sesiones establecidas y se basen en varios parámetros de la máquina a la vez (sistema operativo, procesador, IP, dirección Ethernet, etc.). También monitorizar sistemas DNS, cachés de ARP, *spools* de correo, etc., para detectar cambios en la información que invaliden a anteriores.

- **Ingeniería social**: no es propiamente una cuestión informática, pero también es necesaria para que las personas no empeoren la seguridad. Medidas adecuadas como aumentar la información o educar a usuarios y técnicos en temas de seguridad: controlar qué personal dispondrá de información crítica de seguridad y en qué condiciones puede cederla a otros. Los servicios de ayuda y mantenimiento de una empresa pueden ser un punto crítico: controlar quién posee información de seguridad, y cómo la usa.

  Respecto a usuarios finales, mejorar su cultura de contraseñas, evitar que la dejen apuntada en cualquier sitio, a la vista de terceros, o simplemente la divulguen.

2. Seguridad del sistema

Ante los posibles ataques, tenemos que disponer de mecanismos de prevención, detección y recuperación de nuestros sistemas.

Para la prevención local, hay que examinar los diferentes mecanismos de autentificación y permisos de accesos a los recursos para definirlos correctamente y poder garantizar la confidencialidad y la integridad de nuestra información. En este caso, nos estaremos protegiendo de atacantes que hayan obtenido acceso a nuestro sistema, o bien de usuarios hostiles que quieran saltarse las restricciones impuestas en el sistema.

Respecto a la seguridad en red, tenemos que garantizar que los recursos que ofrecemos (si proporcionamos unos determinados servicios) tienen los parámetros de confidencialidad necesarios y que los servicios no pueden ser usados por terceros no deseados, de modo que, un primer paso será controlar que los servicios ofrecidos sean los que realmente queremos, y que no estamos ofreciendo además otros servicios que no tenemos controlados. En el caso de servicios de los que nosotros somos clientes, también habrá que asegurar los mecanismos de autentificación, en el sentido de que accedemos a los servidores correctos y no existen casos de suplantación de servicios o servidores (normalmente bastante difíciles de detectar).

Respecto a las aplicaciones y a los mismos servicios, además de garantizar la correcta configuración de niveles de acceso mediante permisos y de autentificación de los usuarios permitidos, tendremos que vigilar la posible explotación de *bugs* en el software. Cualquier aplicación, por muy bien diseñada e implementada que esté, puede tener un número más o menos alto de errores que pueden ser aprovechados para, con ciertas técnicas, saltarse las restricciones impuestas. En este caso, practicamos una política de prevención que pasa por mantener el sistema actualizado en la medida de lo posible, de manera que, o bien actualizamos ante cualquier nueva corrección, o bien somos conservadores y mantenemos aquellas versiones que sean más estables en cuestión de seguridad. Normalmente, esto significa verificar periódicamente unos cuantos sitios de seguridad para conocer los últimos fallos detectados en el software y las vulnerabilidades que se derivan de éstos, y que podrían exponer nuestros sistemas a fallos de seguridad, ya sea local o por red.

3. Seguridad local

La seguridad local [Peñ] [Hatb] es básica para la protección del sistema [Deb][Hatc], ya que, normalmente, tras un primer intento de acceso desde la red, es la segunda barrera de protección antes de que un ataque consiga hacerse con parte del control de la máquina. Además, la mayoría de los ataques acaban haciendo uso de recursos internos del sistema.

Nota

Diversos ataques, aunque vengan del exterior, tienen como finalidad conseguir el acceso local.

3.1. *Bootloaders*

Respecto a la seguridad local, ya en arranque se nos pueden presentar problemas debido al acceso físico que un intruso pudiera tener a la máquina.

Uno de los problemas ya se encuentra en el arranque del sistema. Si el sistema puede arrancarse desde disco o CD, un atacante podría acceder a los datos de una partición GNU/Linux (o también Windows) sólo con montar el sistema de ficheros y podría colocarse como usuario *root* sin necesidad de conocer ninguna contraseña. En este caso, se necesita proteger el arranque del sistema desde la BIOS, por ejemplo, protegiendo el acceso por contraseña, de manera que no se permita el arranque desde CD (por ejemplo mediante un LiveCD) o disquete. También es razonable actualizar la BIOS, ya que también puede tener fallos de seguridad. Además, hay que tener cuidado, porque muchos de los fabricantes de BIOS ofrecen contraseñas extras conocidas (una especie de *backdoor*), con lo cual no podemos depender de estas medidas en exclusiva.

El siguiente paso es proteger el *bootloader*, ya sea lilo o grub, para que el atacante no pueda modificar las opciones de arranque del *kernel* o modificar directamente el arranque (caso de grub). Cualquiera de los dos puede protegerse también por contraseñas.

En grub, el fichero /sbin/grub-md5-crypt pide la contraseña y genera una suma md5 asociada. Después, el valor obtenido se introduce en /boot/grub/grub.conf. Bajo la línea *timeout*, se introduce:

```
password --md5 suma-md5-calculada
```

Para lilo se coloca, o bien una contraseña global con:

```
password = contraseña
```

o bien una en la partición que queramos:

```
image = /boot/vmlinuz-version
  password = contraseña
  restricted
```

En este caso *restricted* indica además que no se podrán cambiar los parámetros pasados al *kernel* desde línea de comandos. Hay que tener cuidado de poner el fichero /etc/lilo.conf protegido a sólo lectura/escritura desde el *root* (chmod 600).

Otro tema relacionado con el arranque es la posibilidad de que alguien que tenga acceso al teclado reinicie el sistema, debido a que si se pulsa CTRL+ALT+DEL, se provoca una operación de *shutdown* en la máquina. Este comportamiento viene definido en /etc/inittab, con una línea como:

```
ca:12345:ctrlaltdel:/sbin/shutdown -t1 -a -r now
```

Si se comenta, esta posibilidad de reiniciar quedará desactivada. O por el contrario, puede crearse un fichero /etc/shutdown.allow, que permite a ciertos usuarios poder reiniciar.

3.2. *Passwords* y *shadows*

Las contraseñas típicas de los sistema UNIX iniciales (y de primeras versiones GNU/Linux) estaban encriptadas mediante unos algoritmos DES (pero con claves pequeñas, y una llamada de sistema se encargaba de encriptar y desencriptar, en concreto crypt, ver el man).

Normalmente, se encontraban en el fichero /etc/passwd, en el segundo campo, por ejemplo:

```
user:sndb565sadsd:...
```

Pero el problema está en que este fichero es legible por cualquier usuario, con lo que un atacante podía obtener el fichero y utilizar un ataque de fuerza bruta, hasta que desencriptase los *passwords* contenidos en el fichero, o bien mediante ataques de fuerza bruta por medio de diccionarios.

El primer paso es utilizar los nuevos ficheros /etc/shadow, donde se guarda ahora la contraseña. Este fichero es sólo legible por el *root*, y por nadie más. En este caso, en /etc/passwd aparece un asterisco (*) donde antes estaba la contraseña encriptada. Por defecto, las distribuciones de GNU/Linux actuales usan contraseñas de tipo *shadow* a no ser que se les diga que no las usen.

Un segundo paso es cambiar el sistema de encriptación de las contraseñas por uno más complejo y difícil de romper. Ahora, tanto Fedora como Debian ofrecen contraseñas por md5; normalmente nos dejan escoger el sistema en tiempo de instalación. Hay que tener cuidado con las contraseñas md5, ya que si usamos NIS, podríamos tener algún problema; si no, todos los clientes y servidores usarán md5 para sus contraseñas. Las contraseñas se pueden reconocer en /etc/shadow porque vienen con un prefijo “1”.

Otras posibles actuaciones consisten en obligar a los usuarios a cambiar la contraseña con frecuencia (el comando change puede ser útil), imponer restricciones en el tamaño y el contenido de las contraseñas, y validarlas con diccionarios de términos comunes.

Respecto a las herramientas, es interesante disponer de un *cracker* de contraseñas (o sea, un programa para obtener contraseñas), para comprobar la situación real de seguridad de las cuentas de nuestros usuarios, y forzar así el cambio en las que detectemos inseguras. Dos de las más utilizadas por administradores son John the Ripper y "crack". Pueden funcionar también por diccionario, con lo cual, será interesante disponer de algún ASCII de español (pueden encontrarse por la red). Otra herramienta es "Slurpie", que puede probar varias máquinas a la vez.

Una cuestión que debe tenerse en cuenta siempre es hacer estas pruebas sobre nuestros sistemas. No hay que olvidar que los administradores de otros sistemas (o el proveedor de acceso o ISP) tendrán sistemas de detección de intrusos habilitados y podemos ser objeto de denuncia por intentos de intrusión, ya sea ante las autoridades competentes (unidades de delitos informáticos) o en nuestro ISP para que se nos cierre el acceso. Hay que tener mucho cuidado con el uso de herramientas de seguridad, que están siempre al filo de la navaja entre ser de seguridad o de intrusión.

3.3. Suid y sticky bits

Otro problema importante son algunos permisos especiales que son utilizados sobre ficheros o *script*.

El bit sticky se utiliza sobre todo en directorios temporales, donde queremos que en algunos grupos (a veces no relacionados), cualquier usuario pueda escribir, pero sólo pueda borrar bien el propietario del directorio, o bien el propietario del fichero que esté en el directorio. Un ejemplo clásico de este bit es el directorio temporal /tmp. Hay que vigilar que no haya directorios de este tipo, ya que pueden permitir que cualquiera escriba en ellos, por lo que habrá que comprobar que no haya más que los puramente necesarios como temporales. El bit se coloca mediante (chmod +t dir), y puede quitarse con -t. En un ls aparecerá como un directorio con permisos drwxrwxrwt (fijaos en la última t).

El bit setuid permite a un usuario ejecutar (ya sea un ejecutable o un *shell script*) con los permisos de otro usuario. Esto en algún caso puede ser de utilidad, pero es potencialmente peligroso. Es el caso, por ejemplo, de programas con setuid de root: un usuario, aunque no tiene permisos de root, puede ejecutar un programa con setuid que puede disponer de permisos internos de usuario *root*. Esto es muy peligroso en caso de *scripts*, ya que podrían editarse y modificarse para hacer cualquier cosa. Por lo tanto, hay que tener controlados estos programas, y en caso de que no se necesite el setuid, eliminarlo. El

bit se coloca mediante chmod +s, ya sea aplicándolo al propietario (se llama entonces suid) o al grupo (se llama bit sgid); puede quitarse con -s. En el caso de visualizar con ls, el fichero aparecerá con -rwSrw-rw (fijaos en la S), si es sólo suid, en sgid la S aparecería tras la segunda w.

En caso de utilizar chmod con notación octal, se usan cuatro cifras, donde las tres últimas son los permisos clásicos rwxrwxrwx (recordad que hay que sumar en la cifra 4 para r, 2 w, y 1 para x), y la primera tiene un valor para cada permiso especial que se quiera (que se suman): 4 (para suid), 2 (sgid), y 1 (para sticky).

3.4. Habilitación de *hosts*

En el sistema hay unos cuantos ficheros de configuración especiales que permiten habilitar el acceso a una serie de hosts a algunos servicios de red, pero cuyos errores pueden permitir atacar después la seguridad local. Nos podemos encontrar con:

- .rhosts de usuario: permite que un usuario pueda especificar una serie de máquinas (y usuarios) que pueden usar su cuenta mediante comandos "r" (rsh, rcp...) sin necesidad de introducir la contraseña de la cuenta. Esto es potencialmente peligroso, ya que una mala configuración del usuario podría permitir entrar a usuarios no deseados, o que un atacante (con acceso a la cuenta del usuario) cambiase las direcciones en .rhosts para poder entrar cómodamente sin ningún control. Normalmente, no se tendría que permitir crear estos archivos, e incluso habría que borrarlos completamente y deshabilitar los comandos "r".

- /etc/hosts.equiv: es exactamente lo mismo que los ficheros .rhosts pero a nivel de máquina, especificando qué servicios, qué usuarios y qué grupos pueden acceder sin control de contraseña a los servicios "r". Además, un error como poner en una línea de ese fichero un "+", permite el acceso a "cualquier" máquina. Normalmente, hoy en día tampoco suele existir este fichero, y siempre existe la alternativa del servicio ssh a los "r".

- /etc/hosts.lpd: en el sistema de impresión LPD se utilizaba para poner en las máquinas que podían acceder al sistema de impresión. Hay que tener mucho cuidado, si no estamos sirviendo, de deshabilitar completamente el acceso al sistema, y en caso de que lo estemos haciendo, restringir al máximo las máquinas que realmente hacen uso del mismo. O intentar cambiar a un sistema CUPS o LPRng, que tienen mucho más control sobre los servicios. El sistema LPD es un blanco habitual de ataques de tipo gusano o *buffer overflows*, y están documentados varios *bugs* importantes. Hay que estar atentos si utilizamos este sistema y el fichero hosts.lpd.

3.5. Módulos PAM

Los módulos PAM [Peñ][Mor03] son un método que permite al administrador controlar cómo se realiza el proceso de autentificación de los usuarios para determinadas aplicaciones. Las aplicaciones tienen que haber sido creadas y enlazadas a las bibliotecas PAM. Básicamente, los módulos PAM son un conjunto de bibliotecas compartidas que pueden incorporarse a las aplicaciones como método para controlar la autentificación de sus usuarios. Es más, puede cambiarse el método de autentificación (mediante la configuración de los módulos PAM), sin que sea necesario cambiar la aplicación.

Los módulos PAM (las bibliotecas) suelen estar en el directorio /lib/security (en forma de ficheros objeto cargables dinámicamente). Y la configuración de PAM está presente en el directorio /etc/pam.d, donde aparece un fichero de configuración de PAM por cada aplicación que está usando módulos PAM. Nos encontramos con la configuración de autentificación de aplicaciones y servicios como ssh, de login gráfico de X Window System, como xdm, gdm, kdm, xscreensaver... o, por ejemplo, del login del sistema (la entrada por identificador de usuario y contraseña). En las antiguas versiones de PAM, se utilizaba un archivo (típicamente en /etc/pam.conf), que era donde se leía la configuración PAM si el directorio */etc/pam.d* no existía.

Nota

Para saber más, se puede ver la "The Linux-PAM System Administrators' Guide": http://www.kernel.org/pub/linux/libs/pam/Linux-PAM-html

La línea típica de estos ficheros (en /etc/pam.d) tendría este formato (si se utiliza /etc/pam.conf habría que añadir el servicio a que pertenece como primer campo):

```
module-type control-flag module-path arguments
```

donde se especifica:

a) tipo de módulo: si es un módulo que requiere que el usuario se autentifique (*auth*), o de acceso restringido (*account*); cosas que hay que hacer cuando el usuario entra o sale (*session*); o bien hay que actualizar la contraseña (*password*).

b) flags de control: especifican si es necesario (*required*), si es un requisito previo (*requisite*), si es suficiente (*sufficient*) o si es opcional (*optional*). Ésta es una sintaxis. Hay otra más actual que trabaja en parejas valor y acción.

c) la ruta (*path*) del módulo.

d) argumentos que se pasan al módulo (dependen de cada módulo).

Debido a que algunos servicios necesitan diversas líneas de configuración comunes, hay posibilidad de operaciones de inclusión de definiciones comunes de otros servicios, sólo hay que añadir una línea con:

```
@include servicio
```

Un pequeño ejemplo del uso de módulos PAM (en una distribución Debian), puede ser su uso en el proceso de *login* (se han listado también las líneas incluidas provenientes de otros servicios):

```
auth                                requisite pam_securetty.so
auth                                requisitepam_nologin.so
auth                                requiredpam_env.so
auth                                requiredpam_unix.so nullok
account          required           pam_unix.so
session          required           pam_unix.so
session          optional           pam_lastlog.so
session          optional           pam_motd.so
session          optional           pam_mail.so standard noenv
password         required           pam_unix.so nullok obscure min = 4 max = 8 md5
```

Esto especifica los módulos PAM necesarios para controlar la autentificación de usuarios en el *login* (entrada al sistema). Uno de los módulos pam_unix.so es el que realmente hace la verificación de la contraseña del usuario (viendo ficheros passwd, shadow...).

Otros controlan su sesión para ver cuándo ha entrado por última vez, o guardan cuándo entra y sale (para el comando lastlog), también hay un módulo que se encarga de verificar si el usuario tiene correo por leer (también hay que autentificarse), y otro que controla que cambie la contraseña (si está obligado a hacerlo en el primer *login* que haga) y que tenga de 4 a 8 letras, y puede utilizarse md5 para la encriptación.

En este ejemplo podríamos mejorar la seguridad de los usuarios: el *auth* y el *password* permiten contraseñas de longitud nula: es el argumento nullok del módulo. Esto permitiría tener usuarios con contraseñas vacías (fuente de posibles ataques). Si quitamos este argumento, ya no permitimos contraseñas vacías en el proceso de *login*. Lo mismo puede hacerse en el fichero de configuración de passwd (en este caso, el comando de cambio del *password*), que también presenta el nullok. Otra posible acción es incrementar en ambos ficheros el tamaño máximo de las contraseñas, por ejemplo, con max = 16.

3.6. Alteraciones del sistema

Otro problema puede ser la alteración de comandos o configuraciones básicas del sistema, mediante la introducción de troyanos o *backdoors* en el sistema, por la simple introducción de software que sustituya o modifique ligeramente el comportamiento del software de sistema.

Nota

CERT UNIX checklist: http://www.cert.org/tech_tips/usc20_full.html

Un caso típico es la posibilidad de forzar el *root* para que ejecute comandos falsos de sistema; por ejemplo, si el *root* incluyese el "." en su variable de PATH, esto permitiría la ejecución de comandos desde su directorio actual, lo que habilitaría para colocar archivos que sustituyesen comandos del sistema y que serían ejecutados en primer término antes que los de sistema. El mismo proceso

puede hacerse con un usuario, aunque por ser más limitados sus permisos, puede no afectar tanto al sistema, cuanto más a la propia seguridad del usuario. Otro caso típico es el de las pantallas de *login* falsas, pudiéndose sustituir el típico proceso de *login*, passwd, por un programa falso que almacene las contraseñas introducidas.

En caso de estas alteraciones, será imprescindible usar políticas de auditoría de los cambios, ya sea por medio de cálculo de firmas o sumas (gpg o md5), o bien mediante algún software de control como Tripwire o AIDE. Para los troyanos podemos hacer diferentes tipos de detecciones, o bien utilizar herramientas como *chkrootkit*, si éstos viniesen de la instalación de algún *rootkit* conocido.

Nota

chkrootkit, ver:
http://www.chkrootkit.org

4. SELinux

La seguridad tradicional dentro del sistema se ha basado en las técnicas DAC *(discretionary access control*), donde normalmente cada programa dispone de control completo sobre los accesos a sus recursos. Si un determinado programa (o el usuario permitiéndolo) decide hacer un acceso incorrecto (por ejemplo, dejando datos confidenciales en abierto, ya sea por desconocimiento o por malfuncionamiento). Así en DAC, un usuario tiene completo control sobre los objetos que le pertenecen y los programas que ejecuta. El programa ejecutado dispondrá de los mismos permisos que el usuario que lo está ejecutando. Así, la seguridad del sistema dependerá de las aplicaciones que se estén ejecutando y de las vulnerabilidades que éstas pudiesen tener, o del software malicioso que incluyesen, y en especial afectará a los objetos (otros programas, ficheros, o recursos) a los que el usuario tenga acceso. En el caso del *root*, esto comprometería la seguridad global del sistema.

Por otro lado las técnicas MAC (*mandatory access control*), desarrollan políticas de seguridad (definidas por el administrador) donde el sistema tiene control completo sobre los derechos de acceso que se conceden sobre cada recurso. Por ejemplo, con permisos (de tipo Unix) podemos dar acceso a ficheros, pero mediante políticas MAC tenemos control extra para determinar a qué ficheros explícitamente se permite acceso por parte de un proceso, y qué nivel de acceso queremos conceder. Se fijan contextos, en los cuales se indican en qué situaciones un objeto puede acceder a otro objeto.

SELinux [NSAb] es un componente de tipo MAC reciente incluido en la rama 2.6.x del *kernel*, que las distribuciones van incluyendo progresivamente: Fedora/Red Hat lo traen habilitado por defecto (aunque es posible cambiarlo durante la instalación), y en Debian es un componente opcional.

SELinux implementa políticas de seguridad de tipo MAC, permitiendo disponer de un acceso de permisos más fino que los tradicionales permisos de archivo UNIX. Por ejemplo, el administrador podría permitir que se añadiesen datos a un archivo de *log,* pero no reescribirlo o truncarlo (técnicas utilizadas habitualmente por atacantes para borrar las pistas de sus accesos). En otro ejemplo, podría permitirse que programas de red se enlazaran a un puerto (o puertos) que necesitan, y sin embargo denegar el acceso a otros puertos (por ejemplo, podría ser una técnica que permitiese controlar algunos troyanos o *backdoors*).

SELinux fue desarrollado por la agencia US NSA, con aportaciones de diversas compañías para sistemas UNIX y libres, como Linux y BSD. Se liberó en el año 2000 y desde entonces se ha ido integrando en diferentes distribuciones GNU/Linux.

NOTA

Algunos recursos sobre SELinux:

http://www.redhat.com/docs/manuals/enterprise/RHEL-4-Manual/selinux-guide/

http://www.nsa.gov/selinux/

http://fedora.redhat.com/docs/selinux-faq/

http://selinux.sourceforge.net/

Disponemos en SELinux de un modelo de dominio-tipo, donde cada proceso corre en un denominado contexto de seguridad, y cualquier recurso (fichero, directorio, *socket*, etc.) tiene un tipo asociado con él. Hay un conjunto de reglas que indican qué acciones pueden efectuarse en cada contexto sobre cada tipo. Una

ventaja de este modelo contexto-tipo es que las políticas que se definan pueden ser analizadas (existen herramientas) para determinar qué flujos de información están permitidos, por ejemplo, para detectar posibles vías de ataque, o si la política es lo suficientemente completa para cubrir todos los accesos posibles.

Se dispone de lo que se conoce como la base de datos de políticas de SELinux (*SELinux policy database*) que controla todos los aspectos de SELinux. Determina qué contextos puede utilizar cada programa para ejecutarse, y especifica a qué tipos de cada contexto puede acceder.

En SELlinux cada proceso del sistema tiene un contexto compuesto de tres partes: una identidad, un rol y un dominio. La identidad es el nombre de la cuenta del usuario, o bien system_u para procesos de sistema o user_u si el usuario no dispone de políticas definidas. El rol determina qué contextos están asociados. Por ejemplo user_r no tiene permitido tener el contexto sysadm_t (dominio principal para el administrador de sistema). Así un user_r con identidad user_u no puede obtener de ninguna manera un contexto sysadm_t. Un contexto de seguridad se especifica siempre por esta terna de valores como:

root:sysadm_r:sysadm_t

es el contexto para el administrador del sistema, define su identidad, su rol, y su contexto de seguridad.

Por ejemplo, en una máquina con SELinux activado (una Fedora en este caso) podemos ver con la opción -Z del ps los contextos asociados a los procesos:

ps ax –Z

```
LABEL                            PID    TTY    STAT   TIME   COMMAND
system_u:system_r:init_t         1      ?      Ss     0:00   init
system_u:system_r:kernel_t       2      ?      S      0:00   [migration/0]
system_u:system_r:kernel_t       3      ?      S      0:00   [ksoftirqd/0]
system_u:system_r:kernel_t       4      ?      S      0:00   [watchdog/0]
system_u:system_r:kernel_t       5      ?      S      0:00   [migration/1]
system_u:system_r:kernel_t       6      ?      SN     0:00   [ksoftirqd/1]
system_u:system_r:kernel_t       7      ?      S      0:00   [watchdog/1]
system_u:system_r:syslogd_t      2564   ?      Ss     0:00   syslogd -m 0
system_u:system_r:klogd_t        2567   ?      Ss     0:00   klogd -x
system_u:system_r:irqbalance_t   2579   ?      Ss     0:00   irqbalance
system_u:system_r:portmap_t      2608   ?      Ss     0:00   portmap
system_u:system_r:rpcd_t         2629   ?      Ss     0:00   rpc.statd
user_u:system_r:unconfined_t     4812   ?      Ss     0:00   /usr/libexec/gconfd-2 5
user_u:system_r:unconfined_t     4858   ?      Sl     0:00   gnome-terminal
user_u:system_r:unconfined_t     4861   ?      S      0:00   gnome-pty-helper
user_u:system_r:unconfined_t     4862   pts/0  Ss     0:00   bash
user_u:system_r:unconfined_t     4920   pts/0  S      0:01   gedit
system_u:system_r:rpcd_t         4984   ?      Ss     0:00   rpc.idmapd
system_u:system_r:gpm_t          5029   ?      Ss     0:00   gpm -m /dev/input/mice -t exps2
user_u:system_r:unconfined_t     5184   pts/0  R+     0:00   ps ax -Z
user_u:system_r:unconfined_t     5185   pts/0  D+     0:00   Bash
```

y con ls con la opcion -Z podemos ver los contextos asociados a ficheros y directorios:

```
# ls -Z
drwxr-xr-x josep josep user_u:object_r:user_home_t Desktop
drwxrwxr-x josep josep user_u:object_r:user_home_t proves
-rw-r--r-- josep josep user_u:object_r:user_home_t yum.conf
```

y desde la consola podemos conocer nuestro contexto actual con:

```
$ id -Z
user_u:system_r:unconfined_t
```

Respecto al modo de funcionamiento, SELinux presenta dos modos denominados: *permissive* y *enforcing*. En *permissive* se permiten los accesos no autorizados, pero son auditados en los *logs* correspondientes (normalmente directamente sobre /var/log/messages o bien dependiendo de la distribución con el uso de audit en /var/log/audit/audit.log). En *enforcing* no se permite ningún tipo de acceso que no permitan las políticas definidas. También puede desactivarse SELinux a través de su fichero de configuración (habitualmente en /etc/selinux/config), colocando SELINUX=disabled.

Hay que tener cuidado con la activación y desactivación de SELinux, en especial con el etiquetado de contextos en los ficheros, ya que en periodos en que se active/desactive pueden perderse etiquetas, o no ser simplemente etiquetados. Asimismo, la realización de *backups* del sistema de ficheros tiene que tener en cuenta que se conserven las etiquetas de SELinux.

Otro posible problema a tener en cuenta es el gran número de reglas de política de seguridad que pueden llegar a existir y que pueden provocar limitaciones en el control de los servicios. Ante un tipo determinado de malfuncionamiento, cabe primero determinar que precisamente no sea SELinux el que está impidiendo el funcionamiento, por una limitación demasiado estricta de seguridad (ver apartado de crítica a SELinux), u opciones que no esperábamos tener activadas (pueden requerir cambiar la configuración de los booleanos como veremos).

Con respecto a la política usada, SELinux soporta dos tipos diferentes: *targered* y *strict*. En *targered* la mayoría de procesos operan sin restricciones, y solo servicios (ciertos *daemons*) específicos son puestos en diferentes contextos de seguridad que son confinados a la política de seguridad. En *strict* todos los procesos son asignados a contextos de seguridad, y confinados a políticas definidas, de manera que cualquier acción es controlada por las políticas definidas. En principio éstos son los dos tipos de políticas definidas generalmente, pero la especificación está abierta a incluir más.

Un caso especial de política es la MLS (*multilevel security*), que es una política multinivel de tipo *strict*. La idea es dentro de la misma política definir diferentes niveles de seguridad, los contextos de seguridad tienen asociado un campo

adicional de nivel de acceso. Este tipo de política de seguridad (como MLS) suele ser utilizada en organizaciones gubernamentales y militares, donde hay organizaciones jerárquicas con diferentes niveles de información privilegiada, niveles de acceso general, y capacidades de acción diferentes en cada nivel. Para obtener algunas certificaciones de seguridad, se necesita disponer de políticas de seguridad de este tipo.

Se puede definir qué tipo de política se usará en /etc/selinux/config, variable SELINUXTYPE. La política correspondiente, y su configuración, normalmente se encontrará instalada en los directorios /etc/selinux/SELINUXTYPE/, por ejemplo, suele encontrarse en el subdirectorio policy el fichero binario de la política compilada (que es el que se carga en el kernel, al inicializar SELinux).

4.1. Arquitectura

La arquitectura de SELinux está compuesta de los siguientes componentes:

- Código a nivel *kernel*
- La librería compartida de SELinux
- La política de seguridad (la base de datos)
- Herramientas

Examinamos algunas consideraciones con respecto a cada componente:

- El código de *kernel* monitoriza la actividad del sistema, y asegura que las operaciones solicitadas están autorizadas bajo la configuración de políticas de seguridad de SELinux actual, no permitiendo las operaciones no autorizadas, y normalmente genera entradas en *log* de las operaciones denegadas. El código actualmente se encuentra integrado en los *kernels* 2.6.x, siendo en los anteriores ofrecido como serie de patchs.

- La mayoría de utilidades y componentes SELinux no directamente relacionados con el *kernel*, hacen uso de la librería compartida, llamada libselinux1.so. Que proporciona una API para interaccionar con SELinux.

- La política de seguridad es la que está integrada en la base de datos de reglas de SELinux. Cuando el sistema arranca (con SELinux activado), carga el fichero binario de política, que habitualmente reside en /etc/security/selinux (aunque puede variar según la distribución).

  El fichero binario de políticas se crea a partir de una compilación (vía *make*) de los ficheros fuente de políticas y algunos ficheros de configuración.

Algunas distribuciones (como Fedora) no instalan las fuentes por defecto), las cuales habitualmente podemos encontrar en /etc/security/selinux/src/policy o en /etc/selinux.

Normalmente, estas fuentes consisten en varios grupos de información:

– Los ficheros relacionados con la compilación, *makefile* y *scripts* asociados.

– Ficheros de la configuración inicial, usuarios y roles asociados.

– Ficheros de Type-enforcement, que contienen la mayoría de las sentencias del lenguaje de políticas asociado a un contexto particular. Hay que tener en cuenta que estos ficheros son enormes, típicamente decenas de miles de líneas. Con lo cual puede verse el problema de encontrar bugs o definir cambios en las políticas.

– Y ficheros que sirven para etiquetar los contextos de los ficheros y directorios durante la carga, o en determinados momentos.

• Herramientas: Incluyen comandos usados para administrar y usar SELinux. Versiones modificadas de comandos estandard Linux. Y Herramientas para el análisis de políticas y el desarrollo.

Veamos de este último apartado las herramientas típicas de que solemos disponer:

Algunos de los comandos principales:

Nombre	Utilización
chcon	Etiqueta un fichero especifico, o un conjunto de ficheros con un contexto especifico.
checkpolicy	Realiza diversas acciones relacionadas con las políticas, incluyendo la compilación de las políticas a binario, típicamente se invoca desde las operaciones de makefile.
getenforce	Genera mensaje con el modo actúa de SELinux (*permissive* o *enforcing*). O bien desactivado si es el caso.
getsebool	Obtiene la lista de booleanos, o sea la lista de opciones on/off, para cada contexto asociado a un servicio, u opción general del sistema.
newrole	Permite a un usuario la transición de un rol a otro.
runn_init	Utilizado para activar un servicio (*start, stop*), asegurándose de que se realiza en el mismo contexto que cuando se arranca automáticamente (con init).
setenforce	Cambia de modo a SELinux, `0 permissive`, `1 enforcing.`
setfiles	Etiqueta directorios y subdirectorios con los contextos adecuados, es típicamente utilizado en la configuración inicial de SELinux.
setstatus	Obtiene el estado del sistema con SELinux.

Por otra parte, se modifican algunos comandos GNU/Linux con funcionalidades u opciones nuevas: como por ejemplo cp, mv, install, ls, ps... por ejemplo modificando en los ficheros la etiqueta para asociar el contexto de seguridad (como pudimos comprobar con la opción -Z de ls). Id se modifica para incluir la opción de mostrar el contexto actual del usuario. Y en ps vimos que incluía una opción para visualizar los contextos de seguridad actuales de los procesos.

Además se modifican algunos otros programas comunes para soportar SELinux como:

– cron: Modificado para incluir los contextos para los trabajos en ejecución por cron.
– login: Modificado para que coloque el contexto de seguridad inicial para el usuario cuando entra en el sistema.
– logrotate: Modificado para preservar el contexto de los *logs* cuando estén recopilados.
– pam: Modificado para colocar el contexto inicial del usuario y para usar la API SELinux y obtener acceso privilegiado a la información de *passwords*.
– ssh: Modificado para colocar el contexto inicial del usuario cuando éste entra en el sistema.
– Y varios programas adicionales que modifiquen /etc/passwd o /etc/shadow.

También en algunas distribuciones se incluyen herramientas para la gestión de SELinux, como las setools(-gui), que llevan varias herramientas de gestión y análisis de las políticas. Así como alguna herramienta especifica para controlar los contextos asociados a los diferentes servicios soportados por SELinux en la distribución, por ejemplo la herramienta system-config-securitylevel en Fedora dispone de un apartado para la configuración de SELinux como podemos observar en la siguiente figura:

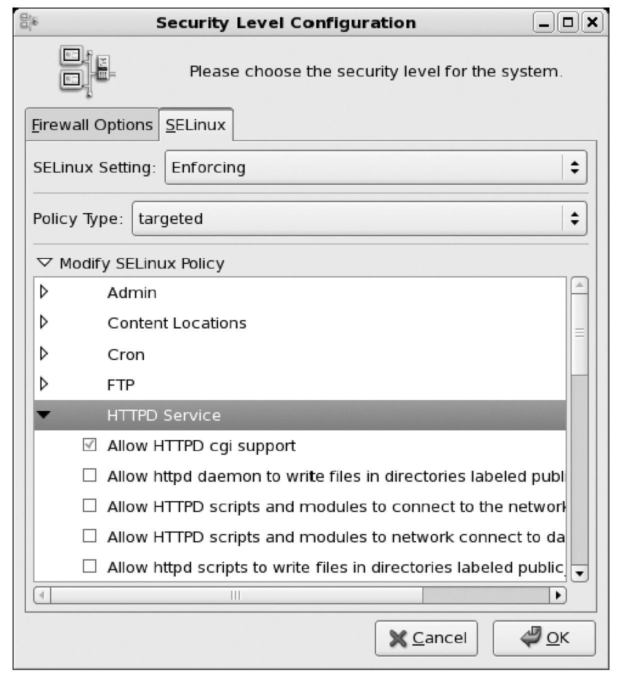


Figura 1. Interfaz en Fedora para configuración SELinux

En la figura se observa la configuración de booleanos para diversos servicios y opciones genéricas, entre ellas el servidor web. También podemos obtener esta lista con el comando getsebool -a, y con los comando setsebool/togglesebool, podemos activar/desactivar las opciones.

En Fedora, por ejemplo, encontramos soporte de booleanos para (entre otros): Cron, ftp, httpd (apache), dns, grub, lilo, nfs, nis, cups, pam, ppd, samba, protecciones contra accesos incorrectos a la memoria de los procesos, etc.

La configuración de booleanos permite la personalización de la política SELinux en tiempo de ejecución. Los booleanos son utilizados como valores condicionales de las reglas de la política usada, esto permite modificaciones de la política sin necesidad de cargar una nueva política.

4.2. Crítica

Algunos administradores, y expertos en seguridad han criticado especialmente a SELinux por ser demasiado complejo para configurar y administrar. Se argumenta que, por su complejidad intrínseca, incluso usuarios experimentados pueden cometer errores, dejando la configuración de SELinux no segura o inservible, y el sistema vulnerable. Aunque esto es discutible hasta cierto punto, ya que aunque tuviésemos SELinux mal configurado, aún seguirían activos los permisos UNIX, SELinux no permitirá una operación que los permisos originales ya no permitían, de hecho podemos verlo como una etapa más de seguridad mas estricta.

También hay factores de rendimiento que pueden afectar, debido al enorme tamaño de las políticas, pueden disminuir las prestaciones, debido al gran uso de memoria y al tiempo inicial de carga, y en algún caso al procesamiento de las reglas. Cabe pensar que estamos hablando prácticamente de un sistema de unas 10.000 reglas de la política. Y aún puede ser un número mayor si escogemos política de tipo *strict,* con la necesidad de especificar absolutamente todas las opciones a controlar. Normalmente el procesamiento por política en formato binario, y el uso de booleanos para inhabilitar reglas, permite un uso más eficiente del sistema.

Otro aspecto que suele molestar a los administradores es el problema adicional de determinar, ante un determinado malfuncionamiento, cuál es el origen o causa inicial. Debido a que es habitual que finalmente nos encontremos que el problema provenía de una configuración excesivamente restrictiva (quizás por desconocimiento del administrador) de SELinux para un determinado servicio.

En última instancia, cabe señalar el amplio soporte para la seguridad que SELinux proporciona, y que como administradores hemos de ser conscientes de las capacidades y peligros de cualquier nueva técnica que utilicemos.

5. Seguridad en red

5.1. Cliente de servicios

Como clientes de servicios, básicamente tenemos que asegurar que no ponemos en peligro a nuestros usuarios (o se pongan en peligro solos) por usar servicios inseguros. Evitar el uso de servicios que no utilicen encriptación de datos y contraseñas (ftp, telnet, correo no seguro). Utilizar en este caso técnicas de conexión encriptada, como SSH y SSL.

Nota

Como clientes de servicios, tendremos que evitar el uso de servicios inseguros.

Otro punto importante se refiere a la posible sustitución de servidores por otros falsos, o bien a técnicas de secuestro de sesiones. En estos casos tenemos que disponer de mecanismos de autentificación potentes, que nos permitan verificar la autenticidad de los servidores (por ejemplo, SSH y SSL tienen algunos de estos mecanismos). Y también tendremos que verificar la red en busca de intrusos que intenten sustituciones de servidores, además de políticas correctas de filtrado de paquetes mediante cortafuegos (*firewalls*), que nos permitan sacar nuestros paquetes de petición y usar los servidores adecuados, controlando los paquetes de entrada que recibamos como respuestas.

5.2. Servidor: inetd y xinetd

La configuración de los servicios de red [Mou01], tal como hemos visto, se realiza desde varios lugares [Ano99][Hat01][Peñ]:

- En /etc/inetd.conf o el equivalente directorio/etc/xinetd.d: estos sistemas son una especie de "superservidores", ya que controlan varios servicios hijos y sus condiciones de arranque. El servicio inetd es utilizado en Debian y xinetd, y en Fedora (en Debian puede ser instalado opcionalmente en sustitución de inetd).

- Servidores iniciados en arranque: según el *runlevel* tendremos una serie de servicios arrancados. El arranque se originará en el directorio asociado al *runlevel*, por ejemplo, en Debian el *runlevel* por defecto es el 2, los servicios arrancados serán arrancados desde el directorio /etc/rc2.d, seguramente con enlaces a los *scripts* contenidos en /etc/init.d, en los cuales se ejecutará con el parámetro start, stop, restart, según corresponda.

- Otros servicios de tipo RPC: asociados a llamadas remotas entre máquinas son, por ejemplo, utilizados en NIS y NFS. Puede examinarse cuáles hay con el comando rpcinfo -p.

Otros ficheros de apoyo (con información útil) son: /etc/services, que está formado por una lista de servicios locales o de red conocidos, junto con el nombre del protocolo (tcp, udp u otros), que se utiliza en el servicio, y el puerto que utiliza; /etc/protocols es una lista de protocolos conocidos; y /etc/rpc es una lista de servidores RPC, junto con los puertos usados. Estos ficheros vienen con la distribución y son una especie de base de datos que utilizan los comandos y herramientas de red para determinar los nombres de servicios, protocolos, o rpc y sus puertos asociados. Cabe destacar que son ficheros más o menos históricos, que no tienen por qué contener todas las definiciones de protocolos y servicios, pueden asimismo buscarse diferentes listas en Internet de puertos conocidos.

Una de las primeras acciones a realizar por el administrador será deshabilitar todos aquellos servicios que no esté utilizando o no tenga previsto utilizar, habrá que documentarse sobre la utilización de los servicios [Mou01], y qué software puede necesitarlos. [Neu]

En el caso de /etc/inetd.conf, sólo tenemos que comentar la línea del servicio que hay que deshabilitar, colocando un "#" como primer carácter en la línea.

En el otro modelo de servicios, utilizado por defecto en Fedora (y opcional en Debian), el xinetd, la configuración reside en el fichero /etc/xinetd.conf, donde se configuran algunos valores por defecto de control de *logs,* y después la configuración de cada servicio hijo se hace a través de un fichero dentro del directorio /etc/xinetd.d. En cada fichero se define la información del servicio, equivalente a la que aparece en el inetd.conf, en este caso, para desactivar un servicio sólo hay que poner una línea disable = yes dentro del fichero del servicio. Xinetd tiene una configuración más flexible que inetd, al separar la configuración de los diferentes servicios en diferentes archivos, y tiene bastantes opciones para limitar las conexiones a un servicio, en el número o la capacidad de éstas; todo lo cual permite un mejor control del servicio y, configurando adecuadamente, podemos evitar algunos de los ataques por denegación de servicio (DoS o DDoS).

Respecto al tratamiento de los servicios de los *runlevel* desde comandos de la distribución, ya mencionamos varias herramientas en la unidad de administración local, que permitían habilitar o deshabilitar servicios. También existen herramientas gráficas como ksysv de KDE, o el system-config-services y ntsysv en Fedora (en Debian son recomendables sysv-rc-conf, rcconf o bum). Y a un nivel inferior, podemos ir al nivel de *runlevel* que queramos (/etc/rcx.d), y desactivar los servicios que se desee cambiando las S o K iniciales del *script* por otro texto: un método sería, por ejemplo: cambiar S20ssh, por STOP_S20ssh, y ya no arrancará; la próxima vez, cuando volvamos a necesitarlo, quitamos el prefijo y lo volvemos a tener activo. O quizás el uso recomendado de utilidades simples para poner, quitar o activar un determinado servicio (como service y chkconfig en Fedora, o similares en Debian como update-rc.d y invoke.rc.d).

Otro aspecto es el cierre de servicios no seguros. Ttradicionalmente, en el mundo UNIX se habían utilizado servicios de transferencia de archivos como ftp, de conexión remota como telnet, y comandos de ejecución (*login* o copia) remotos, muchos de los cuales comenzaban con la letra “r” (por ejemplo, rsh, rcp, rexec...). Otros peligros potenciales son los servicios *finger* y rwhod, que permitían obtener información desde la red de los usuarios de las máquinas; aquí el peligro estaba en la información que podía obtener un atacante y que le podía facilitar mucho el trabajo. Todos estos servicios no deberían ser usados actualmente debido a los peligros potenciales que implican. Respecto al primer grupo:

a) ftp, telnet, en las transmisiones por red no encriptan las contraseñas, y cualquiera puede hacerse con las contraseñas pde servicios o las cuentas asociadas (por ejemplo, mediante un *sniffer*).

b) rsh, rexec, rcp además presentan el problema de que bajo algunas condiciones ni siquiera necesitan contraseñas (por ejemplo, si se hace desde sitios validados en fichero .rhosts), con lo cual vuelven a ser inseguros, y dejan grandes puertas abiertas.

La alternativa es usar clientes y servidores seguros que soporten la encriptación de los mensajes y autentificación de los participantes. Hay alternativas seguras a los servidores clásicos, pero actualmente la solución más usada es mediante la utilización del paquete OpenSSH (que puede combinarse también con OpenSSL para entornos web). OpenSSH ofrece soluciones basadas en los comandos ssh, scp y sftp, permitiendo sustituir a los antiguos clientes y servidores (se utiliza un *daemon* denominado sshd). El comando ssh permite las antiguas funcionalidades de telnet, rlogin, y rsh entre otros, y scp sería el equivalente seguro de rcp, y sftp del ftp.

Respecto a SSH, también hay que tener la precaución de usar ssh version 2. La primera versión tiene algunos *exploits* conocidos; hay que tener cuidado al instalar OpenSSH, y si no necesitamos la primera versión, instalar sólo soporte para ssh2 (ver opción Protocol en fichero de configuración /etc/ssh/ssh_config).

Además, la mayoría de servicios que dejemos activos en nuestras máquinas tendrían después que ser filtrados a través de *firewall*, para asegurar que no fuesen utilizados o atacados por personas a los que no iban destinados.

6. Detección de intrusiones

Con los sistemas de detección de intrusos [Hat01] (IDS) se quiere dar un paso más adelante. Una vez hayamos podido configurar más o menos correctamente nuestra seguridad, el paso siguiente consistirá en una detección y prevención activa de las intrusiones.

Nota

Los sistemas IDS nos permiten la detección a tiempo de intrusos, usando nuestros recursos o explorando nuestros sistemas en busca de fallos de seguridad.

Los sistemas IDS crean procedimientos de escucha y generación de alertas al detectar situaciones sospechosas, o sea, buscamos síntomas de posibles accidentes de seguridad.

Los hay desde sistemas basados en la información local, por ejemplo, recopilan información de los *log* del sistema, vigilan cambios en los ficheros de sistema o bien en las configuraciones de los servicios típicos. Otros sistemas están basados en red e intentan verificar que no se produzcan comportamientos extraños, como por ejemplo los casos de *spoofing*, donde hay falsificaciones de direcciones conocidas; controlar tráfico sospechoso, posibles ataques de denegación de servicio, detectando tráfico excesivo hacia determinados servicios, controlando que no hay interfaces de red en modo promiscuo (síntoma de *sniffers* o capturadores de paquetes).

Ejemplos

Algunos ejemplos de herramientas IDS serían: Logcheck (verificación de logs), TripWire (estado del sistema mediante sumas md5 aplicadas a los archivos), AIDE (una version libre de TripWire), Snort (IDS de verificación de estado de una red completa).

7. Protección mediante filtrado (*wrappers* y *firewalls*)

Los **TCP wrappers** [Mou01] son un software que actúa de intermediario entre las peticiones del usuario de servicio y los *daemons* de los servidores que ofrecen el servicio. Muchas de las distribuciones vienen ya con los *wrappers* activados y podemos configurar los niveles de acceso. Los *wrappers* se suelen utilizar en combinación con inetd o xinetd, de manera que protejan los servicios que ofrecen.

El *wrapper* básicamente sustituye al *daemon* (demonio) del servicio por otro que actúa de intermediario (llamado tcpd, normalmente en /usr/sbin/tcpd). Cuando éste recibe una petición, verifica el usuario y origen de ésta, para determinar si la configuración del *wrapper* del servicio permite o no utilizarlo. Además, incorpora facilidades de generar *logs,* o bien informar por correo de los posibles intentos de acceso y después ejecuta el *daemon* adecuado asignado al servicio.

Por ejemplo, supongamos la siguiente entrada en inetd:

```
finger stream tcp nowait nobody /usr/etc/in.fingerd in.fingerd
```

Ésta se cambia por:

```
finger stream tcp nowait nobody /usr/sbin/tcpd in.fingerd
```

de manera que, cuando llegue una petición, será tratada por el *daemon* tcpd que se encargará de verificar el acceso (para más detalles, ver página man tcpd).

También existe un método alternativo de *wrapper* TCP, que consiste en que la aplicación original se compile con la biblioteca de *wrappers*. Entonces la aplicación no tiene por qué estar en inetd, y podremos controlarla igual que el primer caso con la configuración que comentamos a continuación.

Nota

Los *wrappers* nos permiten controlar la seguridad mediante listas de acceso a nivel de servicios.

El sistema de *wrappers* se controla desde los ficheros /etc/hosts.deny, donde especificamos qué servicios denegamos y a quién, por medio de opciones, como un pequeño *shell* para guardar la información del intento, y el fichero /etc/hosts.allow, donde solemos colocar qué servicio vamos a utilizar, seguido de la lista de a quién dejamos entrar (más adelante, en el taller, veremos un pequeño ejemplo). También existen los comandos tcpdchk, que testan la configuración de los ficheros *hosts* (ver man hosts_access y hosts_options), para comprobar que son correctos, es decir, testa la configuración. El otro comando útil es tcpdmatch, al cual damos un nombre de

servicio y un posible cliente (usuario, y/o *host*), y nos dice qué haría el sistema ante esta situación.

7.1. *Firewalls*

Un *firewall* (o cortafuegos) es un sistema o grupo de sistemas que refuerza las políticas de control de acceso entre redes. El *firewall* puede estar implementado en software, como una aplicación especializada corriendo en un computador individual, o bien puede tratarse de un dispositivo especial dedicado a proteger uno o más computadores.

En general dispondremos, o bien de la aplicación de *firewall* para proteger una máquina concreta conectada directamente a Internet (ya sea directa o por proveedor), o bien podemos colocar en nuestra red una o varias máquinas dedicadas a esta función, de modo que protejan nuestra red interna.

Técnicamente, la mejor solución es disponer de un computador con dos o más tarjetas de red que aíslen las diferentes redes (o segmentos de red) conectadas, de manera que el software de *firewall* en la máquina (o si fuese un hardware especial) se encargue de conectar los paquetes de las redes y determinar cuáles pueden pasar o no, y a qué red.

Este tipo de *firewall* suele combinarse con un *router* para enlazar los paquetes de las diferentes redes. Otra configuración típica es la de *firewall* hacia Internet, por ejemplo con dos tarjetas de red: en una obtenemos/proporcionamos trafico a Internet y en la otra enviamos o proporcionamos el tráfico a nuestra red interna, pudiendo así eliminar el tráfico que no va destinado a nosotros, y también controlar el tráfico que se mueve hacia Internet, por si no queremos que se tenga acceso a algunos protocolos, o bien sospechamos que hay posibilidades de fugas de información por algunos ataques. Una tercera posibilidad es la máquina individual conectada con una única tarjeta de red hacia Internet, directa o bien a través de un proveedor. En este caso, sólo queremos proteger nuestra máquina de ataques de intrusos, de tráfico no deseado o de que se vea comprometida al robo de datos.

Es decir, en estos casos podemos comprobar que el firewall –dependiendo de si es software o no, de si la máquina tiene una o varias tarjetas de red o de si protege a una máquina individual o a una red– puede tener configuraciones y usos diferentes.

El *firewall* en general permite definir al usuario una serie de políticas de acceso (cuáles son las máquinas a las que se puede conectar o las que pueden recibir información y el tipo de información) por medio del control de los puertos TCP/UDP permitidos de entrada (*incomming*) o de salida (*outcomming*). Algunos *firewalls* vienen con políticas preconfiguradas; en algún caso sólo dicen si

Nota

Los *firewall* permiten establecer seguridad a nivel de paquetes y conexiones de comunicación.

se quiere un nivel de seguridad alto, medio o bajo; otros permiten personalizar las opciones totalmente (máquinas, protocolos, puertos, etc.).

Otra técnica a veces relacionada es NAT (*network address translation*). Esta técnica proporciona una vía para ocultar las direcciones IP usadas en la red privada y las oculta de Internet, pero mantiene el acceso desde las máquinas. Uno de los métodos típicos es el denominado *masquerading*. Utilizando NAT masquerading, uno o varios dispositivos en la red pueden aparecer como una única dirección IP vistos desde fuera. Esto permite conectar varios computadores a un único dispositivo de conexión externa; por ejemplo, un caso de *router* ADSL en casa permite conectar varias máquinas sin necesidad de que el proveedor nos proporcione diferentes direcciones IP. Los *routers* ADSL suelen ofrecer algún tipo de NAT *masquerading*, y también posibilidades de *firewall*. Suele ser bastante común utilizar una combinación de ambas técnicas. En este caso, entra en juego, además de la configuración de la máquina del *firewall* (los casos vistos antes), la configuración de la red privada interna que queremos proteger.

7.2. Netfilter: IPtables

El *kernel* Linux (a partir de versiones 2.4.x) proporciona un subsistema de filtrado denominado Netfilter [Net], que proporciona características de filtrado de paquetes y también NAT. Este sistema permite usar diferentes interfaces de filtrado, entre ellas, la más usada se denomina IPtables. El comando principal de control es iptables. Anteriormente [Hata], se proporcionaba otro filtrado denominado ipchains en los *kernel* 2.2 [Gre], el sistema tenía una sintaxis diferente (aunque parecida). En los *kernel* 2.0 se utilizaba otro sistema denominado ipfwadm. Aquí (y en los ejemplos posteriores) vamos a tratar sólo con Netfilter/IPTables (es decir, con *kernels* versiones 2.4/2.6).

Nota

Netfilter, ver: http://www.netfilter.org

Ipchains, ver: http://www.netfilter.org/ipchains/

La interfaz del comando IPtables permite realizar las diferentes tareas de configuración de las reglas que afectan al sistema de filtrado: ya sea generación de *logs,* acciones de pre y post *routing* de paquetes, NAT, y *forwarding* (reenvío) de puertos.

Nota

IPTables aporta diferentes elementos como las tablas, *chains*, y las propias reglas.

Arranque del servicio con: /etc/init.d/iptables start, si no estaba configurado ya en el *runlevel*.

El comando iptables -L lista las reglas activas en ese momento en cada una de las cadenas (*chains*). Si no se han configurado previamente, suelen ser por defecto aceptar todos los paquetes de las cadenas (o *chains*) de *input* (entrada), *output* (salida) y *forward* (reenvío).

El sistema de IPTables tiene como nivel superior las tablas. Cada una contiene diferentes cadenas, que a su vez contienen diferentes reglas. Las tres tablas que existen son: Filter, NAT y Mangled. La primera sirve para las propias normas

de filtrado, la segunda para realizar traslación de direcciones dentro de un sistema que utilice NAT, y la tercera, menos usada, sirve para especificar algunas opciones de control de los paquetes, y cómo gestionarlos. Concretamente, si estamos con un sistema directamente conectado a Internet, utilizaremos, en general, tan sólo la tabla Filter. Si el sistema está en una red privada que tiene que pasar por un *router*, *gateway* o *proxy* (o combinación de éstos), seguramente dispondremos de un sistema de NAT o IP *masquerading*; si estamos configurando precisamente la máquina que permite acceso externo, tendremos que tocar la tabla NAT y la Filter. Si la máquina está en un sistema de red privada, pero es una de las máquinas internas, será suficiente con la tabla Filter, a no ser que sea un servidor el que haga NAT a otro segmento de red.

Si un paquete llega al sistema, en el *firewall* se mirará primero si existen reglas en la tabla NAT, por si hay que hacer traducciones de direcciones hacia la red interna (las direcciones normalmente no son visibles hacia fuera); después se mirarán las reglas de la tabla Filter para decidir si se van a dejar pasar los paquetes, o si no son para nosotros, y tenemos reglas *forward* para saber hacia dónde los reenviamos. Por el contrario, cuando nuestros procesos generan paquetes, las reglas *output* de la tabla Filter controlan si los dejamos salir o no, y si hubiese sistema NAT, las reglas efectuarían la traducción de direcciones de manera que se enmascarasen. En la tabla NAT suele haber dos cadenas: prerouting y postrouting. En la primera, las reglas han de decidir si hay que hacer algún *routing* del paquete y cuál será la dirección de destino. En el segundo, se decide finalmente si el paquete se pasa o no hacia el interior (la red privada, por ejemplo). Y también existe una cadena *output* para el tráfico que se genere localmente de salida a la red privada, ya que prerouting no lo controla (para más detalles, podéis ver man iptables).

A continuación comentaremos algunos aspectos y ejemplos de configuración de la tabla Filter (para las otras tablas, se puede consultar la bibliografía asociada).

La configuración típica de la tabla Filter es de una serie de reglas que especifican qué se hace dentro de una determinada cadena (o *chain*), como las tres anteriores (*input, output* o *forward*). Normalmente, se especifica:

```
iptables -A chain -j target
```

donde *chain* es *input, output o forward,* y *target* el destino que se le va a dar al paquete que se corresponda con la regla. La opción -A añade la regla a las existentes. Con esto hay que tener cuidado, ya que el orden importa. Hay que colocar las menos restrictivas al principio, puesto que, si primero ponemos una regla que elimine los paquetes, aunque haya otra regla, ésta no será tenida en cuenta. La opción -j permite decidir qué haremos con los paquetes, típicamente *accept* ('aceptar'), *reject* ('rechazar'), o *drop* (simplemente 'perderlo'). Es importante la diferencia entre *reject* y *drop*. Con el primero, rechazamos el paquete y normalmente informamos al emisario de que hemos rechazado el intento de conexión (normalmente por un paquete de tipo ICMP). Con el se-

gundo (drop), simplemente perdemos el paquete como si nunca hubiese existido y no enviamos ningún tipo de respuesta. Otro *target* utilizado es *log,* para enviar el paquete al sistema de log. Normalmente, en este caso hay dos reglas, una con el *log* y otra igual con *accep*t, *drop* y *reject*, para permitir enviar al *log* la información de paquetes aceptados, rechazados o perdidos.

Al poner la regla, también puede usarse la opción -I (insertar) para indicar una posición, por ejemplo:

```
iptables -I INPUT 3 -s 10.0.0.0/8 -j ACCEPT
```

que nos dice que se coloque la regla en la cadena *input* en tercera posición; y se van a aceptar paquetes (-j) que provengan (con fuente, o *source*, -s) de la subred 10.0.0.0 con netmask 255.0.0.0. Con -D de forma parecida podemos borrar o un número de regla, o la regla exacta, como se especifica a continuación, borrando la primera regla de la cadena o la regla que mencionamos:

```
iptables -D INPUT 1
iptables -D INPUT -s 10.0.0.0/8 -j ACCEPT
```

También hay reglas que permiten definir una “política” por defecto de los paquetes (opción -P); se va a hacer con todos los paquetes lo mismo. Por ejemplo, se suele decir que se pierdan todos los paquetes por defecto, y se habilitan luego los que interesan, y muchas veces también se evita que haya *forwarding* de paquetes si no es necesario (si no actuamos de *router*), esto podría ponerse:

```
iptables -P INPUT DENY
iptables -P OUTPUT REJECT
iptables -P FORWARD REJECT
```

Todo lo cual establece unas políticas por defecto que consisten en denegar la entrada de paquetes, no permitir salir y no reenviar paquetes. Ahora se podrán añadir las reglas que conciernen a los paquetes que deseemos utilizar, diciendo qué protocolos, puertos y orígenes o destinos queremos permitir o evitar. Esto puede ser difícil, ya que tenemos que conocer todos los puertos y protocolos que utilicen nuestro software o servicios. Otra táctica sería dejar sólo activos aquellos servicios que sean imprescindibles y habilitar con el *firewall* el acceso de los servicios a las máquinas deseadas.

Algunos ejemplos de estas reglas de la tabla Filter podrían ser:

```
1)iptables -A INPUT -s 10.0.0.0/8 -d 192.168.1.2 -j DROP
2)iptables -A INPUT -p tcp --dport 113 -j REJECT --reject-with tcp-reset
3)iptables -I INPUT -p tcp --dport 113 -s 10.0.0.0/8 -j ACCEPT
```

Donde:

1) Perdemos los paquetes que vengan de 10.x.x.x con destino a 192.168.1.2.

2) Rechazamos los paquetes tcp con destino al puerto 113, emitiendo una respuesta de tipo tcp-reset.

3) Los mismos paquetes que en 2) pero que provengan de 10.x.x.x serán aceptados.

Respecto a los nombres de protocolos y puertos, el sistema iptables utiliza la información proporcionada por los ficheros /etc/services y /etc/protocols, pudiéndose especificar la información (de puerto y protocolo) de forma numérica o bien por nombre (hay que tener cuidado en este caso de que la información de los ficheros sea correcta, que no hayan sido modificados, por ejemplo, por un atacante).

La configuración de iptables suele establecerse mediante llamadas consecutivas al comando iptables con las reglas. Esto crea un estado de reglas activas que pueden consultarse con iptables -L, si deseamos salvarlas para que sean permanentes, podemos hacerlo en Fedora con:

```
/etc/init.d/iptables save
```

Y se guardan en:

```
/etc/sysconfig/iptables
```

En Debian puede hacerse:

```
/etc/init.d/iptables save nombre-reglas
```

Hay que tener cuidado de que exista previamente el directorio /var/log/iptables, que es donde se guardan los ficheros; nombre-reglas será un fichero en el directorio.

Con (/etc/init.d/iptables load) podemos cargar las reglas (en Debian hay que dar el nombre del fichero de reglas), aunque Debian soporta unos nombres por defecto de ficheros, que son *active* para las reglas normales (las que se usarán en un *start* del servicio) e *inactive* para las que quedarán cuando se desactive el servicio (se haga un *stop*). Otra aproximación comúnmente usada es la de colocar las reglas en un fichero *script* con las llamadas iptables que se necesiten y llamarlas por ejemplo colocándolas en el *runlevel* necesario, o con enlace hacia el *script* en /etc/init.d.

7.3. Paquetes de *firewalls* en las distribuciones

Respecto a herramientas de configuración más o menos automáticas del *firewall*, existen varias posibilidades, pero hay que tener en cuenta que no suelen ofrecer las mismas prestaciones que la configuración manual de iptables (que en la mayoría de casos sería el proceso recomendado). Algunas herramientas son:

- **lokkit**: en Fedora/Red Hat, muy básico, sólo permite elegir al usuario un nivel de seguridad que desea (alto, medio o bajo). Después enseña los ser-

vicios que se ven afectados y podemos dejar, o no, pasar al servicio cambiando la configuración por defecto. El mecanismo utilizado por debajo es iptables. Puede verse la configuración final de reglas que realiza /etc/sysconfig/iptables que, a su vez, es leído por el servicio iptables, que se carga en arranque, o bien por parada o arranque mediante /etc/init.d/iptables con las opciones start o stop. En Debian también es posible instalarlo, pero deja la configuración de las reglas en /etc/defaults/lokkit-l, y un *script* en /etc/init.d/lokkit-l. También existe una versión gráfica llamada gnome-lokkit.

- **Bastille** [Proa]: programa de seguridad bastante completo y didáctico, ya que nos explica paso a paso diferentes recomendaciones de seguridad y si las queremos aplicar, así como la configuración del *firewall* (el programa es interactiveBastille). Funciona en varias distribuciones, tanto en Fedora como en Debian.

- **fwbuilder:** una herramienta que permite construir las reglas del *firewall* de forma gráfica. Se puede utilizar en varios operativos (GNU/Linux tanto Fedora como Debian, OpenBSD, MacOS), con diferentes tipos de *firewalls* (iptables incluido).

Nota

Ver:
http://www.fwbuilder.org/

- **firestarter**: una herramienta gráfica (Gnome) para la creación del *firewall*. Es muy completa, prácticamente manejando todas las posibilidades de iptables, pero asimismo, dispone de asistentes que facilitan la construcción intuitiva del *firewall*. Dispone además de un monitor en tiempo real, para detectar las intrusiones.

Normalmente, cada uno de estos paquetes utiliza un sistema de reglas que guarda en algún fichero propio de configuración, y que suele arrancar como servicio o como ejecución de *script* en el *runlevel* por defecto.

7.4. Consideraciones finales

Aunque dispongamos de *firewalls* bien configurados, hay que tener presente que no son una medida de seguridad absoluta, ya que hay ataques complejos que pueden saltarse el control, o bien falsear datos que creen confusión. Además, las necesidades de conectividad modernas obligan a veces a crear software que permita el *bypass* (cruce) de los *firewalls*:

Nota

Nunca se debe confiar en un único mecanismo o sistema de seguridad. Hay que establecer la seguridad del sistema a diferentes niveles.

- Tecnologías como IPP, protocolo de impresión utilizado por CUPS, o el WebDAV, protocolo de autoría y actualización de sitios web, permiten pasar (o es necesario que pasen) las configuraciones de los *firewalls*.

- A menudo se utiliza (por ejemplo, los anteriores protocolos y otros) una técnica denominada *tunneling*, que básicamente encapsula protocolos no

permitidos, sobre la base de otros que sí que lo están; por ejemplo, si un *firewall* permite sólo paso de tráfico http (puerto 80 por defecto), es posible escribir un cliente y un servidor (cada uno a un lado diferente del *firewall*) que hablen cualquier protocolo conocido por ambos, pero que en la red es transformado en un *protocolo* http estándar, con lo cual, el tráfico puede cruzar el *firewall*.

- Los códigos móviles por web (ActiveX, Java, y JavaScript), cruzan los *firewalls*, y por lo tanto es difícil proteger los sistemas si éstos son vulnerables a los ataques contra agujeros descubiertos.

Así, aunque los *firewalls* son una muy buena solución a la mayor parte de la seguridad, siempre pueden tener vulnerabilidades y dejar pasar tráfico que se considere válido, pero que incluya otras fuentes posibles de ataques o vulnerabilidades. En seguridad nunca hay que considerar (y confiar en) una única solución, y esperar que nos proteja absolutamente; hay que examinar los diversos problemas, plantear soluciones que los detecten a tiempo y políticas de prevención que nos curen en salud, por lo que pueda pasar.

8. Herramientas de seguridad

Algunas de las herramientas pueden considerarse también herramientas de ataque. Por lo tanto, se recomienda probar estas herramientas sobre máquinas de nuestra red local o privada; nunca hacerlo con IP de terceros, ya que éstos podrían tomarlo como intrusiones y pedirnos responsabilidades, o a nuestro ISP, o avisar a las autoridades competentes para que nos investiguen o cierren nuestro acceso.

A continuación, comentamos brevemente algunas herramientas, así como algunos usos que pueden utilizar:

a) TripWire: esta herramienta mantiene una base de datos de sumas de comprobación de los ficheros importantes del sistema.

Puede servir como sistema IDS preventivo. Nos sirve para "tomar" una foto del sistema, poder comparar después cualquier modificación introducida y comprobar que éste no haya sido corrompido por un atacante. El objetivo aquí es proteger los archivos de la propia máquina, evitar que se produzcan cambios como, por ejemplo, los que podría haber provocado un *rootkit*. Así, cuando volvamos a ejecutar la herramienta, podemos comprobar los cambios respecto a la ejecución anterior. Hay que elegir un subconjunto de archivos que sean importantes en el sistema, o fuentes de posibles ataques. TripWire es propietario, pero hay una herramienta libre equivalente llamada AIDE.

b) Nmap [Insb]: es una herramienta de escaneo de puertos para redes grandes. Puede escanear desde máquinas individuales a segmentos de red. Permite diversos modelos de scan, dependiendo de las protecciones que tenga el sistema. También tiene técnicas que permiten determinar el sistema operativo que usan las máquinas remotas. Puede emplear diferentes paquetes de TCP y UDP para probar las conexiones. Existe una interfaz gráfica denominada xnmap.

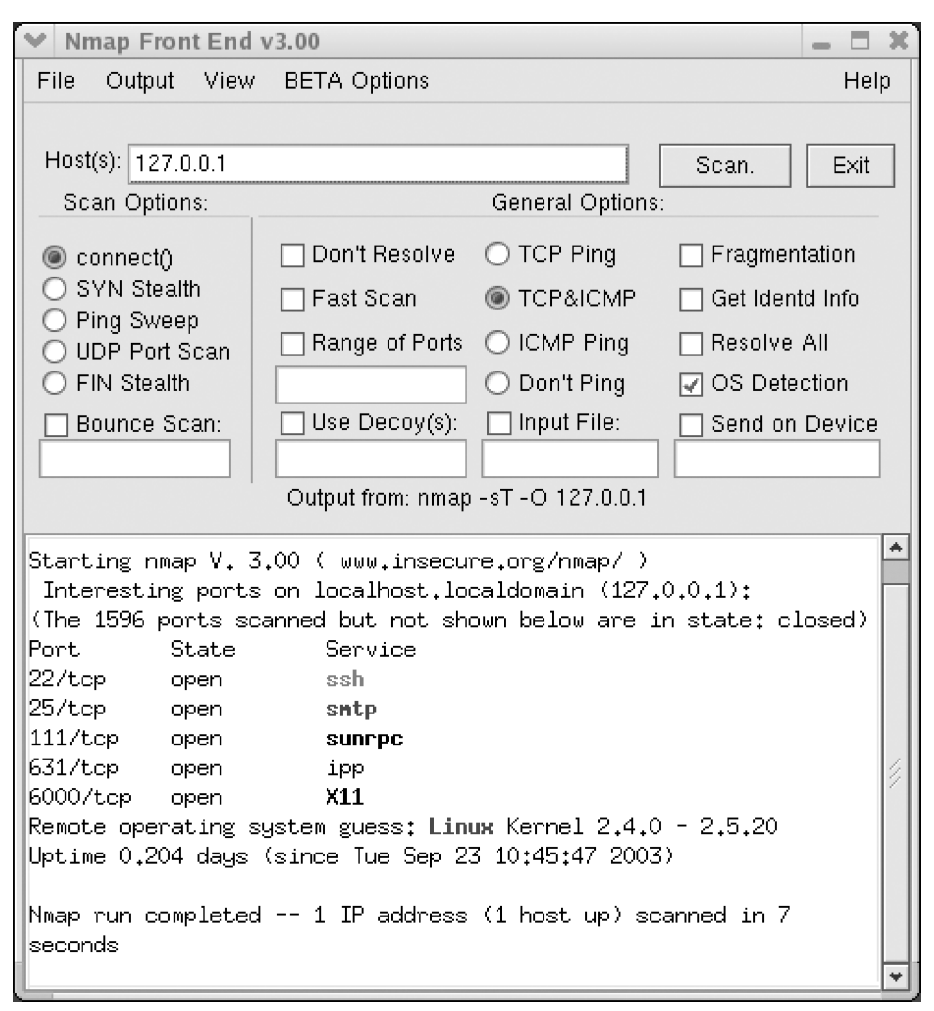


Figura 2. xnmap analizando servicios locales

c) Wireshark [Wir] (antes llamado Ethereal): es un analizador de protocolos y captura el tráfico de la red (actua de *sniffer*). Permite visualizar el tráfico capturado, ver estadísticas y datos de los paquetes individuales, y agruparlos, ya sea por origen, destino, puertos o protocolo. Puede incluso reconstruir el tráfico de una sesión entera de un protocolo TCP.

d) Snort [Sno]: es un sistema IDS que permite realizar análisis de tráfico en tiempo real y guardar *logs* de los mensajes. Permite realizar análisis de los protocolos, búsquedas por patrones (protocolo, origen, destino, etc.). Puede ser usado para detectar diversos tipos de ataques. Básicamente, analiza el tráfico de la red para detectar patrones que puedan corresponder a un ataque. El sistema utiliza una serie de reglas para, o bien anotar la situación (*log*) o bien producir un aviso (*alert*) o descartar la información (*drop*).

e) Nessus [Nes]: es un detector de vulnerabilidades conocidas (probando diferentes técnicas de intrusión) y asesora sobre cómo mejorar la seguridad para las detectadas. Es un programa modular que incluye una serie de *plugins* (más de 11.000) para los diferentes análisis. Utiliza una arquitectura cliente-servidor, con un cliente gráfico que muestra los resultados y el servidor que

realiza las diferentes comprobaciones sobre las máquinas. Tiene capacidad para examinar redes enteras. Genera informes de resultados en forma de *reports* que pueden exportarse a diferentes formatos (por ejemplo, HTML). Hasta el 2005, Nessus 2 era una herramienta libre, pero la compañía decidió convertirla en propietaria en la versión Nessus 3. Todavía en GNU/Linux, se sigue utilizando Nessus 2, que sigue con licencia GPL, y una serie de *plugins* que se van actualizando. Nessus 3 como herramienta propietaria para GNU/Linux, es más potente y es ampliamente utilizada, siendo una de las herramientas más populares de seguridad, normalmente se mantiene libre de coste una versión con los *plugins* menos actualizados que la versión de pago.

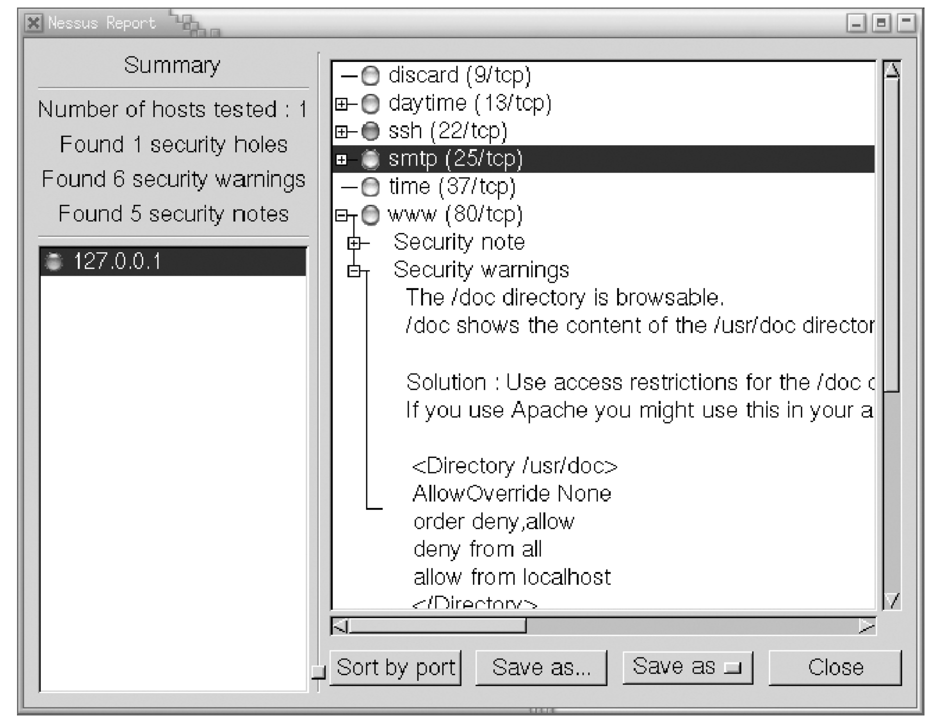


Figura 3. Cliente de Nessus mostrando el informe de vulnerabilidades y posibles soluciones

Podemos encontrar muchas más herramientas de seguridad disponibles. Un buen lugar a examinar es http://sectools.org, donde los creadores del Nmap mantienen una lista de herramientas populares votadas por los usuarios.

9. Análisis logs

Observando los ficheros de *log* [Ano99][Fri02], podemos hacernos una idea rápida del estado global del sistema, así como de las últimas ocurrencias, y detectar accesos (o intentos) indebidos. Pero también cabe tener presente que los *logs*, si ha sucedido una intrusión real, pueden haber sido limpiados o falseados. La mayoría de archivos de *log* residen en el directorio /var/log.

Muchos de los servicios pueden poseer *logs* propios, que normalmente se establecen en la configuración (mediante el correspondiente fichero de configuración). La mayoría suelen utilizar las facilidades de *log* incorporadas en el sistema Sys *log,* mediante el demonio Syslogd. Su configuración reside en /etc/syslog.conf. Esta configuración suele establecerse por niveles de mensajes: existen diferentes tipos de mensajes según su importancia. Normalmente suelen aparecer niveles debug, info, err, notice, warning, err, crit, alert, emerg, en que más o menos ésta sería la ordenación de importancia de los mensajes (de menor a mayor). Normalmente, la mayoría de los mensajes se dirigen al *log* /var/log/messages, pero puede definirse que cada tipo vaya a ficheros diferentes y también se puede identificar quién los ha originado; típicamente el *kernel*, correo, *news*, el sistema de autentificación, etc.

Por lo tanto, cabe examinar (y en todo caso adaptar) la configuración de Syslog para conocer en qué *logs* podemos encontrar/ generar la información. Otro punto importante es controlar su crecimiento, ya que según los que estén activos y las operaciones (y servicios) que se realicen en el sistema, los *logs* pueden crecer bastante. En Debian y Fedora se controla a través de logrotated, un demonio que se encarga periódicamente de hacer copias y comprimirlas de los *logs* más antiguos; se puede encontrar su configuración general en /etc/logrotate.conf, algunas aplicaciones hacen configuraciones específicas que se pueden encontrar en el directorio /etc/logrotate.d.

Comentamos en los siguientes puntos algunos ficheros de *log* que habría que tener en cuenta (quizás los más utilizados):

a) /var/log/messages: es el fichero de *log* por defecto del demonio Syslogd, pero habría que revisar su configuración, no sea que esté cambiado a otro sitio, o haya varios. Este fichero contiene una amplia variedad de mensajes de orígenes diversos (diferentes demonios, servicios o el mismo *kernel*); todo lo que se observase anómalo tendría que ser verificado. En caso de que haya habido una intrusión, mirar alrededor de la fecha de la intrusión.

b) /var/log/utmp: este fichero contiene información binaria para cada usuario que está actualmente activo. Es interesante para determinar quién

está dentro del sistema. El comando who utiliza este fichero para proporcionar esta información.

c) /var/log/wtmp: cada vez que un usuario entra en el sistema, sale, o la máquina reinicia, se guarda una entrada en este fichero. Es un fichero binario de donde el comando last obtiene información, en que se menciona qué usuarios entraron o salieron, cuándo y dónde se originó la conexión. Puede ser útil para buscar dónde (en qué cuentas) se originó la intrusión o detectar usos de cuentas sospechosas. También existe una variación del comando llamada lastb, que lista los intentos de *login* que no pudieron validarse correctamente y se usa el fichero /var/log/btmp (puede ser necesario crearlo si no existe). Estos mismos fallos de autentificación también suelen enviarse al log auth.log. De modo parecido, el comando lastlog utiliza otro fichero /var/loglastlog para verificar cuál fue la última conexión de cada uno de los usuarios.

d) /var/log/secure: suelen utilizarse en Fedora para enviar los mensajes de los tcp *wrappers* (o los *firewalls*). Cada vez que se establece una conexión a un servicio de inetd, o bien en el caso de Red Hat 9, para xinetd (con su propia seguridad) se añade un mensaje de *log* a este fichero. Pueden buscarse intentos de intrusos de servicios que no suelen usarse, o bien máquinas no familiares que se intentan conectar.

En el sistema de *logs*, otra cosa que habría que asegurar es que el directorio de *logs* /var/ *log,* pueda ser sólo escribible por el *root* (o *deamons* asociados a servicios). De otro modo, cualquier atacante podría falsificar la información de los *logs*. Aun así, si el atacante consigue acceso a *root*, puede, y suele, borrar las pista de sus accesos.

10. Taller: análisis de la seguridad mediante herramientas

Vamos a realizar algunos de los procesos descritos sobre un sistema Debian, para configurar mejor su seguridad.

Primero examinaremos qué ofrece nuestra máquina a la red. Para ello, utilizaremos la herramienta nmap como escaneador de puertos. Con el comando (desde *root*):

```
nmap -sTU -O localhost
```

obtenemos:

```
root@maquina:˜# nmap -sUT -O localhost
starting nmap 3.27 ( www.insecure.org/nmap/ ) at 2003-09-17 11:31 CEST
Interesting ports on localhost (127.0.0.1):
```

(The 3079 ports scanned but not shown below are in state: closed)

Port	State	Service
9/tcp	open	discard
9/udp	open	discard
13/tcp	Open	daytime
22/tcp	Open	ssh
25/tcp	open	smtp
37/tcp	open	time
37/udp	open	time
80/tcp	open	http
111/tcp	open	sunrpc
111/udp	open	sunrpc
113/tcp	open	auth
631/tcp	open	ipp
728/udp	open	unknown
731/udp	open	netviewdm3
734/tcp	open	unknown

```
Remote operating system guess: Linux kernel 2.4.0-2.5.20
Uptime 2.011 days (since Mon Sep 15 11:14:57 2003)
Nmap run completed --1 IP address (1 host up) scanned in 9.404 seconds
```

Podemos observar que nos ha detectado un gran número de servicios abiertos (dependiendo de la máquina podría haber más: telnet, ftp, finger...), tanto en protocolos tcp como udp. Algunos servicios como discard, daytime, time pueden ser útiles en alguna ocasión, pero normalmente no tendrían que estar abiertos a red, ya que se consideran inseguros. SMTP es el servicio de reenvío,

y *routing*, el de correo; si actuamos como *host* o servidor de correo, tendría que estar activo; pero si sólo leemos y escribimos correo mediante cuentas POP3 o IMAP, no tiene por qué estarlo.

Otra manera de detectar servicios activos sería mediante la búsqueda de puertos activos a la escucha, esto puede hacerse con netstat -lut.

El servicio nmap también puede aplicarse con el nombre DNS o IP de la máquina, así, vemos lo que se vería desde el exterior (con localhost vemos lo que puede ver la propia máquina), o mejor incluso podríamos utilizar una máquina de una red externa (por ejemplo, un PC cualquiera conectado a Internet), para examinar lo que verían de nuestra máquina desde fuera.

Vamos ahora a ir a /etc/inetd.conf para desactivar estos servicios. Buscamos líneas como:

```
discard stream tcp nowait root internal
smtp stream tcp nowait mail /usr/sbin/exim exim -bs
```

y les colocamos un # al principio de la línea (sólo a aquellos servicios que queramos desactivar y sepamos qué hacen realmente (consultar páginas *man*), o se recomiende su desactivación. Otro caso de desactivación recomendado sería el de los servicios de ftp, telnet, finger... y usar ssh para sustituirlos.

Ahora tenemos que reiniciar inetd para que vuelva a leer la configuración que hemos cambiado: /etc/init.d/inetd restart.

Volvemos a nmap:

22/tcp	open	ssh
80/tcp	open	http
111/tcp	open	sunrpc
111/udp	open	sunrpc
113/tcp	open	auth
631/tcp	open	ipp
728/udp	open	unknown
734/tcp	open	unknown

De lo que nos queda, tenemos el servicio ssh, que queremos dejar, y el servidor de web, que lo pararemos de momento:

```
/etc/init.d/apache stop
```

ipp es el servicio de impresión asociado a CUPS. En administración local vimos que había una interfaz web de CUPS que se conectaba al puerto 631. Si queremos tener una idea de a qué se dedica un puerto determinado, podemos mirar en /etc/services:

```
root@maquina:# grep 631 /etc/services
ipp 631/tcp            # Internet Printing Protocol
ipp 631/udp            # Internet Printing Protocol
```

Si no estamos actuando como servidor de impresión hacia el exterior, tenemos que ir a la configuración de CUPS y eliminar esa prestación (por ejemplo, colocando un *listen* 127.0.0.1:631, para que sólo escuche la máquina local), o limitar el acceso a las máquinas permitidas.

Nos aparecen también algunos puertos como desconocidos, en este caso el 728 y 734; esto indica que el sistema no ha podido determinar qué nmap está asociado al puerto. Vamos a intentar verlo nosotros. Para ello, ejecutamos sobre el sistema el comando netstat, que ofrece diferentes estadísticas del sistema de red, desde paquetes enviados y recibidos, y errores, hasta lo que nos interesa, que son las conexiones abiertas y quién las usa. Vamos a intentar buscar quién está usando los puertos desconocidos:

```
root@maquina:˜# netstat -anp | grep 728
udp 0 0 0.0.0.0:728 0.0.0.0:* 552/rpc.statd
```

Y si hacemos lo mismo con el 734, observamos también que quien ha abierto el puerto ha sido rpc.statd, que es un *daemon* asociado a NFS (en este caso el sistema tiene un servidor NFS). Si hacemos este mismo proceso con los puertos 111 que aparecían como sunrpc, observaremos que el *daemon* que hay detrás es portmap, que se usa en el sistema de llamadas RPC. El sistema de llamadas RPC (*remote procedure call*) permite utilizar el mecanismo de llamadas remotas entre dos procesos que están en diferentes máquinas. portmap es un *daemon* que se encarga de traducir las llamadas que le llegan por el puerto a los números de servicios RPC internos que se tengan, y es utilizado por diferentes servidores como NFS, NIS, NIS+.

Los servicios RPC que se ofrecen se pueden ver con el comando rpcinfo:

```
root@maquina:˜# rpcinfo -p
programa vers  proto   puerto
100000 2 tcp   111     portmapper
100000 2 udp   111     portmapper
100024 1 udp   731     status
100024 1 tcp   734     status
391002 1 tcp   39797   sgi_fam
391002 2 tcp   39797   sgi_fam
```

donde observamos los servicios RPC con algunos de los puertos que ya se habían detectado. Otro comando que puede resultar útil es lsof, que, entre otras funciones, permite relacionar puertos con los servicios que los han abierto (por ejemplo: lsof -i | grep 731).

El *daemon* portmap es un poco crítico con la seguridad, ya que, en principio, no ofrece mecanismos de autentificación del cliente, pues se supone que se delegan en el servicio (NFS, NIS...). Por lo tanto, portmap podría ser víctima de intentos de DoS que podrían provocar errores en los servicios o hacerlos caer. Normalmente, protegeremos portmap por medio de algún tipo de *wrapper* y/o *firewall*. Si no utilizamos, y no tenemos previsto utilizar, servicios NFS y NIS, lo mejor es desactivar completamente portmap, quitándolo del *runlevel* donde se active. También podemos pararlos momentáneamente con los *scripts* (en Debian):

```
/etc/init.d/nfs-common
/etc/init.d/nfs-kernel-server
/etc/init.d/portmap
```

dándoles el parámetro *stop* para parar los servicios RPC (en este caso NFS).

A continuación, controlaremos la seguridad en base a un *wrapper* sencillo. Vamos a suponer que queremos dejar paso a través de ssh de una máquina determinada, llamémosla 1.2.3.4 (dirección IP). Vamos a cerrar portmap al exterior, ya que no tenemos NIS, y de NFS tenemos servidor pero no estamos sirviendo nada (podríamos cerrarlo, pero lo dejaremos para futuros usos). Vamos hacer un *wrapper* (suponemos que los TCP *wrappers* están ya instalados), modificando los ficheros hosts.deny -j allow. En /etc/hosts.deny:

```
ALL : ALL : spawn (/usr/sbin/safe_finger -l @%h \
| /usr/bin/mail -s "%c FAILED ACCESS TO %d!!" root) &
```

estamos denegando todos los servicios (cuidado, aquellos relacionados con inetd) (primer all) a todos (all), y la acción por tomar será averiguar quién ha pedido el servicio y en qué máquina, y vamos a enviar un correo al *root* informando del intento. Podríamos también escribir un fichero de *log*... Ahora, en /etc/hosts.allow:

```
sshd: 1.2.3.4
```

habilitamos el acceso por la máquina IP 1.2.3.4 en el servidor sshd (del ssh). También podríamos colocar el acceso a portmap, sólo haría falta una línea portmap: la_ip. Podemos colocar una lista de máquinas o bien subredes que puedan utilizar el servicio (ver man hosts.allow). Recordar que también tenemos los comandos `tcpdchk`, para comprobar que la configuración del *wrapper* esté correcta, y `tcpdmatch` para simular qué pasaría con un determinado intento, por ejemplo:

```
root@maquina:˜# tcpdmatch sshd 1.2.3.4

warning: sshd: no such process name in /etc/inetd.conf client: hostname
maquina.dominio.es
client: address 1.2.3.4
server: process sshd
matched: /etc/hosts.allow line 13
access: grantedv
```

nos dice que sería concedido el acceso. Un detalle es que nos dice que sshd no está en el inetd.conf, y si lo verificamos, vemos que así es: no se activa por el servidor inetd, sino por *daemon* en el *runlevel* que estemos. Además (en Debian), éste es un caso de demonio que está compilado con las bibliotecas de *wrappers* incluidas (y, por lo tanto, no necesita tcpd para funcionar). En Debian hay varios *daemons* así: ssh, portmap, in.talk, rpc.statd, rpc.mountd, entre otros. Esto permite asegurar estos *daemons* mediante *wrappers*.

Otra cuestión por verificar son las conexiones actuales existentes. Con el comando netstat -utp, podemos listar las conexiones tcp, udp establecidas con el exterior, ya sean entrantes o salientes; así, en cualquier momento podemos detectar los clientes conectados y a quién estamos conectados. Otro comando importante (de múltiples funciones) es lsof, que puede relacionar ficheros abiertos con procesos o conexiones por red establecidas mediante lsof -i, pudiéndose así detectar accesos indebidos a ficheros.

También podríamos utilizar un *firewall* para procesos similares (o bien como mecanismo añadido). Comenzaremos viendo cómo están las reglas del *firewall* en este momento: (comando iptables -L)

```
root@aopcjj:~# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source              destination
Chain FORWARD (policy ACCEPT)
target prot opt source              destination
Chain OUTPUT (policy ACCEPT)
target prot opt source              destination
```

O sea, que el *firewall* no está colocando ninguna restricción en este momento, y permite la entrada, salida y reenvío de todos los paquetes.

En este punto, podríamos añadir un *firewall* que nos permitiese una gestión más adecuada de paquetes que recibimos y enviamos, y que sería un control previo para mejorar la seguridad. Dependiendo de nuestras necesidades, estableceríamos las reglas necesarias de modo parecido a las que comentamos en los ejemplos de *firewalls* de la unidad.

En caso de poner los *firewalls*, podemos considerar si usar este mecanismo como único y quitar los *wrappers*: podría hacerse, ya que los *firewalls* (en este caso mediante iptables) ofrecen un mecanismo muy potente que nos permite seguir a un paquete por tipo, por protocolo y por lo que está haciendo en el sistema. Un buen *firewall* podría ser más o menos suficiente, pero, por si acaso, más medidas de seguridad no vienen mal. Y en caso de que el *firewall* no estuviese bien diseñado y dejase escapar algunos paquetes, el *wrapper* sería la medida, a nivel de servicio, para parar los accesos no deseados. Por poner una metáfora que se suele utilizar, si nos planteásemos nuestro sistema como la defensa de un castillo medieval, el foso y primeras murallas serían el *firewall*, y la segunda muralla de contención, el *wrapper*.

Actividades

1) Supongamos que colocamos en nuestra máquina un sitio web, por ejemplo con Apache. Nuestro sitio está pensado para diez usuarios internos, pero no controlamos este número. Más adelante nos planteamos poner en Internet este sistema, ya que creemos que puede ser útil para los clientes, y lo único que hacemos es poner el sistema con una IP pública en Internet. ¿Qué tipo de ataques podría sufrir este sistema?

2) ¿Cómo podemos detectar los ficheros con suid en nuestro sistema? ¿Qué comandos serán necesarios? ¿Y los directorios con SUID o SGID? ¿Por qué es necesario, por ejemplo, que /usr/bin/passwd tenga bit de SUID?

3) Los ficheros .rhosts, como hemos visto, son un peligro importante para la seguridad. ¿Podríamos utilizar algún método automático que comprobase su existencia periódicamente? ¿Cómo?

4) Supongamos que queremos deshabilitar un servicio del cual sabemos que tiene el *script* /etc/init.d/servicio que lo controla: queremos desactivarlo en todos los *runlevels* en los que se presenta. ¿Cómo encontramos los *runlevels* en los que está? (por ejemplo, buscando enlaces al *script*).

5) Examinar los servicios en activo en vuestra máquina. ¿Son todos necesarios? ¿Cómo habría que protegerlos o desactivarlos?

6) Practicar el uso de algunas de las herramientas de seguridad descritas (nmap, nessus, etc.).

7) ¿Qué reglas IPtables serían necesarias para una máquina en la que sólo queremos acceso por SSH desde una dirección concreta?

8) ¿Y si queremos únicamente un acceso al servidor web?

Otras fuentes de referencia e información

[Deb] [Hatc] Los sitios de seguridad de las distribuciones.

[Peñ] Imprescindible para Debian, muy buena descripción para seguir paso a paso la configuración de seguridad, [Hatb] sería equivalente para Fedora/Red Hat.

[Mou01] Excelente referencia de seguridad para Red Hat (aplicable también para Debian).

[Hat01] Libros seguridad GNU/Linux, cubriendo amplios aspectos y técnicas.

[Line] Pequeña guía (2 páginas) de seguridad.

[Sei] Guía paso a paso identificando los puntos clave que hay que verificar y los problemas que puedan surgir.

[Net] Proyecto Netfilter, y IPTables.

[Ian] Una lista de puertos TCP/IP.

[Proa] [Sno] [Insb] [Nes] Algunas de las herramientas de seguridad más usadas.

[NSAb] Versión de Linux con miras a la seguridad, producida por la NSA. Referencia para SELinux.

[CERa][Aus][Insa][Incb] [NSAa] Sitios de organismos de seguridad.

[CERb][Ins][San] Vulnerabilidades y *exploits* de los diferentes operativos.

[NSAa][FBI][USA] Algunas “policías” de cibercrimen en Estados Unidos.